COMPLETED 14

ORIGINAL

NUREG/CR-4800
UCID-20679

SIGPI: A User's Manual for Fast Computation of the Probabilistic Performance of Complex Systems

Prepared by C. J. Patenaude

Lawrence Livermore National Laboratory

Prepared for
U.S. Nuclear Regulatory
Commission

BEST COPY AVAILABLE

NOTICE

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, or any of their employees, makes any warranty, expressed or implied, or assumes any legal liability of responsibility for any third party's use, or the results of such use, of any information, apparatus, product or process disclosed in this report, or represents that its use by such third party would not infringe privately owned rights.

NOTICE

Availability of Reference Materials Cited in NRC Publications

Most documents cited in NRC publications will be available from one of the following sources:

1. The NRC Public Document Room, 1717 H Street, N.W.
   Washington, DC 20555
2. The Superintendent of Documents, U.S. Government Printing Office, Post Office Box 37082,
   Washington, DC 20013-7082
3. The National Technical Information Service, Springfield, VA 22161

Although the listing that follows represents the majority of documents cited in NRC publications, it is not intended to be exhaustive.

Referenced documents available for inspection and copying for a fee from the NRC Public Document Room include NRC correspondence and internal NRC memoranda; NRC Office of Inspection and Enforcement bulletins, circulars, information notices, inspection and investigation notices; Licensee Event Reports; vendor reports and correspondence; Commission papers; and applicant and licensee documents and correspondence.

The following documents in the NUREG series are available for purchase from the GPO Sales Program: formal NRC staff and contractor reports, NRC-sponsored conference proceedings, and NRC booklets and brochures. Also available are Regulatory Guides, NRC regulations in the *Code of Federal Regulations*, and *Nuclear Regulatory Commission Issuances*.

Documents available from the National Technical Information Service include NUREG-series reports and technical reports prepared by other federal agencies and reports prepared by the Atomic Energy Commission, forerunner agency to the Nuclear Regulatory Commission.

Documents available from public and special technical libraries include all open literature items, such as books, journal and periodical articles, and transactions. *Federal Register* notices, federal and state legislation, and congressional reports can usually be obtained from these libraries.

Documents such as theses, dissertations, foreign reports and translations, and non-NRC conference proceedings are available for purchase from the organization sponsoring the publication cited.

Single copies of NRC draft reports are available free, to the extent of supply, upon written request to the Division of Information Support Services, Distribution Section, U.S. Nuclear Regulatory Commission, Washington, DC 20555.

Copies of industry codes and standards used in a substantive manner in the NRC regulatory process are maintained at the NRC Library, 7920 Norfolk Avenue, Bethesda, Maryland, and are available there for reference use by the public. Codes and standards are usually copyrighted and may be purchased from the originating organization or, if they are American National Standards, from the American National Standards Institute, 1430 Broadway, New York, NY 10018.

BEST COPY AVAILABLE

NUREG/CR-4800
UCID-20679
RG

SIGPI: A User's Manual for Fast Computation of the Probabilistic Performance of Complex Systems

Manuscript Completed: September 1986
Date Published: May 1987

Prepared by
C. J. Patenaude

Lawrence Livermore National Laboratory
7000 East Avenue
Livermore, CA 94550

Prepared for
Division of Reactor System Safety
Office of Nuclear Regulatory Research
U.S. Nuclear Regulatory Commission
Washington, DC 20555
NRC FIN A0392

BEST COPY AVAILABLE

BLANK PAGE

ABSTRACT

The SIGPI program computes the probability of complex systems as defined by cut sets or other binary product sets. The SIGPI program uses two fast complementary methods of computing the probabilistic performance of complex systems: the Π method and the Σ method.

The Π algorithm exploits the fact that carefully defined system components are often statistically independent conditional to the environment in which they are embedded. The SIGPI program accepts multiple probability data sets to define the various conditional environments and then computes a separate probability value for each environment without having to recompute the entire problem. When conditioning will not allow all components to be treated independently, statistically dependent components may be mixed with the conditioned independent components.

The Σ algorithm computes the probability of combinations of components produced by the Π algorithm by disjointing and partitioning such components, thereby allowing the exact computation of performance. The computational complexity of the overall process is usually a polynomial function of the number of components. A desired accuracy can be specified, and the SIGPI program will halt when that accuracy has been reached.

The program assumes input of up to three data types: cut set data in disjoint normal form, basic component probabilities for independent basic components, and/or mean and covariance data for statistically dependent basic components. The SIGPI program is written in FORTRAN 77 and will be available at the National Energy Software Center (NESC) at Argonne National Laboratory, Argonne, Illinois.

BEST COPY AVAILABLE

BLANK PAGE

Table of Contents

Introduction....1

1. OVERVIEW OF THE ΣΠ ALGORITHM....2
   1.1. The Π Method....2
   1.2. The Σ Method....3
      1.2.1. The Disjointing Algorithm....3
      1.2.2. The Partitioning Algorithm....3
      1.2.3. The Bounding Algorithm....4
   1.3. Using SIGPI....4

2. MATHEMATICS OF ΣΠ....5
   2.1. The Problem....5
   2.2. Disjointing the Matrix....6
   2.3. Partitioning the Matrix....7
   2.4. Conditional Data Sets....8
   2.5. Dependent Data Sets....9

3. SIGPI USAGE....13
   3.1. Running the SIGPI program....13
      3.1.1. Input Data Set File....14
      3.1.2. Probability Data Files....15
         3.1.2.1. Conditionally Independent Data....15
         3.1.2.2. Dependent Data....16
      3.1.3. Output Data....17
      3.1.4. Accuracy Determination....17
      3.1.5. Conditioning Control....17

4. PROGRAM IMPLEMENTATION....18
   4.1. Program Structure....18
      4.1.1. The Control Program....19
      4.1.2. The Preprocessor....19
      4.1.3. The Main Processor....19
   4.2. Data Management....20
      4.2.1. Basic Component Data....21
      4.2.2. The Cut Set Matrix....21
      4.2.3. The Partitioning Block Tree....21
      4.2.4. Subproblem Temporary Storage Area....2?
   4.3. Implementation of the ΣΠ Algorithm....2?
      4.3.1. Definition of the Σ Algorithm....27
      4.3.2. The Partitioning Algorithm....30
      4.3.3. Optimum Component for Disjointing....31
      4.3.4. The System Probability Estimate....32
   4.4. Example of The Σ Algorithm....35

References....40

BEST COPY AVAILABLE

APPENDIX A - Moving SIGPI to Other Computers.....41
A.1. Compile Modules.....41
A.2. Parameters.....51
A.3. Overlays.....52
A.4. Word Size Considerations.....56
A.5. Real Variable Considerations.....56
A.6. Boolean Operations.....56
A.7. Library Functions.....57

APPENDIX B - Test data.....59
B.1. Cut Set Data.....59
B.1.1. Small Coherent Data Set.....59
B.1.2. Small Noncoherent Data Set.....59
B.1.3. Large Coherent Data Set.....60
B.1.4. Large Noncoherent Data Set.....60
B.2. Independent Probability Data.....61
B.3. Mean and Covariance Data Set.....63
B.4. Computer Runs Using the Data Sets.....64
B.4.1. Small set - coherent, independent data.....64
B.4.2. Small set - noncoherent, independent data.....64
B.4.3. Small set - coherent, dependent data.....64
B.4.4. Small set - noncoherent, dependent data.....64
B.4.5. Large set - coherent independent data.....65
B.4.6. Large set - noncoherent independent data.....65
B.4.7. Large set - coherent dependent data.....65
B.4.8. Large set - noncoherent dependent data.....65

EST COPY AVAILABLE

Introduction

The operation of complex systems often involves a variety of uncertainties; consequently, performance is difficult to describe or predict with the accuracy required. Various methods have been developed over the past ten years to account for these uncertainties and their impact on system performance. In the analysis of reliability, security, safety, and risk, the techniques of fault trees, graphs, and networks have been applied to obtain practical methods. Calculating the probabilistic performance of systems described by these methods, however, has not always been computationally possible.

This user's manual describes SIGPI, a program developed at Lawrence Livermore National Laboratory to perform fast computations of probabilistic performance of complex systems. The algorithms of SIGPI were first developed in 1982 and have been applied in the years following to evaluate responses to both environmental threats and to seismic events [Refs. 1-3]. These applications helped expand the method to its present formulation. This version of SIGPI, modified to run under Fortran 77, was sponsored by the U.S. Nuclear Regulatory Commission.

This manual is organized into four sections. Section 1, Overview of the ΣΠ Algorithm, describes the algorithm, briefly explaining both the sigma and pi methods which give the program its name, SIGPI, and make possible the computation of probabilistic performance.

Section 2, Mathematics of ΣΠ, gives an overview of the mathematics of the ΣΠ algorithms and states the problem, explaining the disjointing and partitioning of the matrix, conditional data sets, and statistically dependent data sets. This section can be used to determine if SIGPI will meet your needs.

Section 3, Program Usage, explains how to operate SIGPI for those who already have the program on their systems. This section explains how to run SIGPI, including inputting data set files, probability data files, accuracy determination, and conditioning control.

Section 4, Program Implementation, explains how to implement the program onto a system on which it has not run before. Section 4 explains the program structure, data management, and implementation of the algorithm. Examples are given.

Appendix A gives detailed information about the compile modules and the overlay modules and explains possible modifications to move SIGPI to other computers. Appendix B contains test data.

BEST COPY AVAILABLE

1. OVERVIEW OF THE ΣΠ ALGORITHM

The SIGPI program [Ref. 1] uses two fast complementary methods of computing the probabilistic performance of complex systems: the Π method and the Σ method. The Π method exploits the fact that carefully defined system components are often statistically independent conditional to the environment in which they are embedded, a very convenient fact from a computational point of view. The Σ method computes the probability of combinations of components produced by the Π method by disjointing the components, thereby allowing the exact computation of performance. The computational complexity of the overall process is a polynomial function of the number of components. When statistical independence is not possible, dependent data can be used in the calculation, although both speed and accuracy may suffer. For super-large problems, where costs of precise answers may be prohibitive, a desired accuracy can be specified, and the ΣΠ algorithms will halt when the specified accuracy has been reached.

1.1. The Π Method

The Π method is intended to exploit the structure present when components or subsystems are statistically dependent. Whereas statistically independent components are relatively easy to analyze, the problems with dependent components often grow exponentially or combinatorially with the number of components. When the dependence is due to common-cause or common-mode effects, however, dependent components often become conditionally independent, given the common-cause condition.

The Π method allows the user to fix the common-cause condition, computing the joint distribution of the (now independent) components of interest for that fixed condition by simply multiplying the conditional marginal probabilities of the components, hence the Π name for the method. When a fixed number of common-cause conditions is considered, this computation must be done for each condition. Repeating part of the computation is usually a small price to pay for the simplicity gained. Conditioning can also be used to advantage to lessen the amount of dependence among components.

The ΣΠ algorithm is valid for combinations of independent components, conditionally independent components, and dependent components. For a more thorough discussion of conditional independence, see Refs. 1, 2.

BEST COPY AVAILABLE

1.2. The Σ Method

The Σ method is intimately complementary to the Π method, and is intended to exploit the algebraic structure of random components or components of interest. A recurring operation in vulnerability and reliability analysis is to compute the probability P(A OR B OR ...) for components A, B, ... For the simple case of two components A and B, we get the answer directly with the "inclusion-exclusion" method, P(A OR B) = P(A) + P(B) - P(A AND B). Most current procedures use this formula in one form or another. Because of the need to keep subtracting and adding the errors due to the "AND" terms, however, this method is combinatorial (sometimes worse than exponential) in its complexity as more components are considered.

The Σ method consists of a combination of three techniques: partitioning of the sets, disjointing of the sets, and tight bounding of the probability calculation.

1.2.1. The Disjointing Algorithm

The disjointing algorithm was developed earlier as the STOP algorithm [Ref. 3]. STOP is a fast method of disjointing binary product sets. Components are disjointed by computing B-A, the difference between B and A, thereby allowing the simple form P(A OR B) = P(A) + P(B-A). This principle is then extended to the general case by decomposing the components into pieces whose probabilities can be summed, hence the Σ name for the method. The STOP algorithm represents a set-theoretic viewpoint where a holistic approach is taken in which all cut sets are examined and transformed simultaneously via a "divide and conquer" approach to obtain the disjoint form.

1.2.2. The Partitioning Algorithm

The partitioning portion of the algorithm uses the basic structure inherent in most systems. The technique consists of breaking or partitioning a problem into weakly interacting subproblems, which can then be disjointed. The partitioning algorithm uses DeMorgan's theorem, P(A OR B) = P[NOT (NOT-A AND NOT-B)], to "glue" the subproblems back together after they have been disjointed. During partitioning A and B can represent subsystems as well as individual components as long as the subsystems have no components in common, and no components in A are dependent on components in B.

BEST COPY AVAILABLE

1.2.3. The Bounding Algorithm

The partitioning and disjointing proceed simultaneously, so that each technique can take advantage of the progress of the other. The bounding portion of the algorithm uses a variety of methods to calculate the probabilities of the subproblems while they are being generated. SIGPI also maintains a tight upper and lower bound estimation. The algorithm allows the probability calculation to proceed to any level of accuracy desired within the limits of the computer. When the desired accuracy is reached, the upper and lower bounds are printed, and a new set of conditional probability data is applied to the partitioned and disjointed pieces, taking advantage of the work already accomplished (the Π method). All conditional probability data sets are evaluated, returning to the Σ algorithm as needed.

1.3. Using SIGPI

Several available programs will generate cut sets in a disjunctive normal form (simple sum of products). The SIGPI program will accept cut sets or other binary product sets as input along with multiple probability data sets. SIGPI will then generate multiple sets of upper and lower bounds of the system probability.

BEST COPY AVAILABLE

2. MATHEMATICS OF ΣΠ

This section describes the type of problems that can be solved using SIGPI and describes how they are solved. Terms are defined and techniques are discussed as they apply to SIGPI.

2.1. The Problem

A system of binary product sets can be viewed as a matrix of simple sets as seen in Figure 1, where C_j represents a component of a system, and S_i represents a simple set. E_{ij} represents the value of component j in simple set i and assumes the values [0,1,-], where "0" represents a failed component, "1" represents a working component, and "-" represents a "don't care" condition.

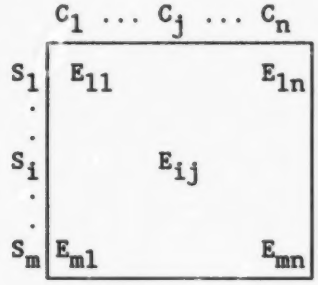


Figure 1. Matrix representation of a collection of m simple sets of n components.

The matrix may be used to represent a system that fails if the simple set i fails, in which case the simple sets are also called cut sets and the matrix is the system failure matrix. Simple set i occurs if all components j are in the state specified by the corresponding entry E_{ij} of the failure matrix.

SIGPI is designed to work with either coherent systems or noncoherent systems. In a coherent system, E_{ij} can assume one of two values [0,-], representing a system that fails when certain combinations of components fail, those which equal 0. In a noncoherent system, on the other hand, at least one of its cut sets occurs only if at least one component of the system is operating properly. Such a system thus fails, dependent on

BEST COPY AVAILABLE

certain components being in a working state. E_{ij} can now assume one of three values [0,1,-].

The probability of failure of a system such as we have described is

$$\text{Probability(system failure)} = P(\bigcup_{i=1}^{m} S_i)$$

where $\bigcup$ signifies the union operator. The simple sets are rarely disjoint. Therefore

$$P(\bigcup_{i=1}^{m} S_i) \neq \sum_{i=1}^{m} P(S_i),$$

and it is natural to seek some transformation of the collection $(S_i: i=1,2,\ldots,m)$ into another collection $(T_j: j=1,2,\ldots,m')$ of simple sets such that T_j are disjoint, and such that

$$P(\bigcup_{i=1}^{m} S_i) = P(\bigcup_{j=1}^{m'} T_j) = \sum_{j=1}^{m'} P(T_j).$$

2.2. Disjointing the Matrix

Since the simple sets so defined can be manipulated by Boolean algebra, we will make use of the following relationships. If X is the universal set and A and B are any two sets, then

$$A \bigcup X = X$$
$$A \bigcap X = A,$$

where $\bigcup$ signifies the union operator and $\bigcap$ signifies the intersection operator. Furthermore, for any collection of $(A_i: i=1,2,\ldots,n)$ of sets

$$\bigcup_{i=1}^{n} (B \bigcap A_i) = B \bigcap (\bigcup_{i=1}^{n} A_i).$$

We also have the expansion operation, where "+" denotes disjoint union and -B is the complement of B.

$$A = (A \bigcap B) + (A \bigcap -B).$$

The containment (or absorption) operation is

$$A = A \bigcup (A \bigcap B).$$

BEST COPY AVAILABLE

The Σ algorithm uses these relationships to factor the simple set input into disjoint sets. Using the containment and expansion operations, the entire matrix structure is exploited by operating on both the component and the set indices. The initial problem is decomposed into two disjoint and simpler problems along a component chosen to minimize the size of both offspring problems. This assures that these two problems are collectively much easier to solve than the parent problem.

The method used to calculate the probability while the cut sets are being disjointed includes keeping a partial probability with each subset. As each subset is decomposed into two new subsets along a component, that component's contribution to the probability of the subset can be calculated. This partial probability is used to calculate an upper and lower bound for each subproblem within the system. For each pass on the whole system, after all of the current subproblems have been processed, these bounds can be used to calculate bounds of the entire system.

2.3. Partitioning the Matrix

The partitioning part of the algorithm is based on the fact that components tend to be clustered in the cut sets. In addition to disjointing the cut sets, this clustering or blocking is searched for. Referring to the trivial example in Figure 2, a block in matrix *M* is a submatrix surrounded only by "don't cares" in the parent matrix, except in the diagonals. More specifically, *M'* is a block in *M* if each of the components which have a "non-don't care" value in any of the simple sets in *M'*, have a "don't care" value in all of the simple sets in *M* which are not in *M'*. These simple sets and components are not necessarily contiguous.

M =

	C_1	C_2	C_3	C_4
S_1	1	0	-	-
	M'			
S_2	0	-	-	-
S_3	-	-	0	1
			M"	
S_4	-	-	-	0

Figure 2. Matrix *M* has block structure consisting of submatrices *M'* and *M"*.

BEST COPY AVAILABLE

The partitioning of the matrix into submatrices is a key step in the Σ algorithm because the disjointing process can increase exponentially with the number of components in the matrix. This partitioning allows the disjointing algorithm to operate on sufficiently small matrices to make the time required to solve the problem a polynomial expression rather than an exponential expression.

Weak interactions always remain because none of the partitioned blocks is disjoint from any other. The partitioned blocks have no components in common, and they must meet the condition of mutual independence although each block may have dependencies among components. This allows the use of DeMorgan's theorem in calculating the probability. The probability of the matrix *M* is the probability of submatrix *M'* "OR" *M"*. Since *M'* and *M"* have no components in common, and the components are conditionally independent, then:

$$P_M = 1.- (1.-P_{M'})(1.-P_{M"}).$$

Often key components prevent the block structure from being utilized. When this occurs, the block is first sent to the disjointing algorithm to be factored into disjoint submatrices. When the troublesome components have been factored from the matrix, the partitioning algorithm is again applied.

2.4. Conditional Data Sets

The disjointing and partitioning process is continued until the algorithm is satisfied. This may happen one of two ways: either a collection of matrices is obtained, each matrix of which consists of only one simple set that is disjoint or partitioned from any other, or the upper and lower bounds are within the desired accuracy range. The current bounds are then written out, and the next probability data set is loaded. All elements of the disjointed and partitioned pieces are preserved, so that the new probability values can be inserted into the problem, and an estimate made for the new conditional data set. If these bounds are within the desired accuracy range, the new upper and lower bounds are written out. If not, the problem with the new data set is returned to the Σ algorithm for more processing. This process continues until all conditional data sets have been processed to the desired accuracy range.

BEST COPY AVAILABLE

2.5. Dependent Data Sets

Whereas the assumption that basic components are independent is reasonable for most problems, there are systems for which such an assumption leads to unacceptable errors. When a cut set $C=(E_1,E_2,\ldots E_n)$ contains dependent basic components, its probability of occurrence $P(C)$ cannot be evaluated from the probabilities $P(E_i)$ of its constituent basic components E_i by the convenient equation

$$P(C) = \sum_{i=1}^{n} P(E_i).$$

Rather, subsets of dependent variables must be evaluated *jointly*, by computing the joint probability of such subsets using their joint distribution function. If the analysis is done in the Boolean domain of component failure or non-failure, the evaluation is combinatorially complex, thus prohibitive.

A more reasonable approach is to take one step back to the causes of component failure, to analyze the origin of the failure probabilities.

The failure of each component is determined by the difference between the strength T of the component and the stress S imposed on the component. The strain U resulting from S is thus

$$U = S - T.$$

When a failure level L is defined, the component fails if $U \geq L$. (Degradation or partial failures are not considered.)

The failure of each basic component E is determined by such a stress/strength relationship, and the probability of failure of the component is thus simply

$$P(E) = P(U \geq L) = P(S - T \geq L).$$

If we consider a cut set C with *independent* components E_i, $i=1,2,\ldots n$, we thus get

$$P(C) = \prod P(E_i) = \prod P(S_i - T_i \geq L_i).$$

Since the variables S_i and T_i are typically continuous random variables, we obtain the product of integrals

BEST COPY AVAILABLE

$$P(C) = \prod_{i=1}^{n} \int_{L_i}^{\infty} (S_i - T_i)\, P(S_i, T_i)\, ds\ dt.$$

When the S_i's or T_i's are *dependent*, however, this product formulation is invalid, and we must compute the n-dimensional integral

$$P(C) = \int_{L_1} \int_{L_2} \cdots \int_{L_n} ((S_1-T_1),(S_2-T_2),\ldots(S_n-T_n))\ P((S_1,T_1),\ldots(S_n,T_n))\ ds\ dt\ \ldots ds\ dt.$$

Because this is an impossible task for arbitrary distributions, we assume that the variables S_i and T_i are multivariate Gaussian, and that all the L_i's equal zero. This *normality* assumption is reasonable from a modeling point of view, and absolutely compelling from a computational point of view. Even with this assumption, significant difficulties exist in integrating more that ten dependent variables because computation times increase exponentially with dimension n.

For statistically dependent components, their joint distribution must be multivariate Gaussian. The user must supply the mean value for each component and the covariance matrix. The integral

$$\frac{|V|^{\frac{1}{2}}}{(2\pi)^{\frac{1}{2}n}} \int_{LL1}^{UL1} \int_{LL2}^{UL2} \cdots \int_{LLn}^{ULn} e^{-\frac{1}{2}(Y-u)'V^{-1}(Y-u)}\, dy_1, dy_2, \ldots dy_n$$

is used to evaluate the probability of a simple set of n components, where Y is the vector of integral variables y_i. V is the covariance matrix of the n components with determinant $|V|$; Y' signifies the transpose of Y. For the computation of the probability of failure of a component (i.e., those components determined to be "0"), SIGPI integrates from a lower limit LLi of $\mu - k\sigma$ to an upper limit ULi of zero, where μ is the mean of component i, σ is the standard deviation of component i and k is some constant chosen to reduce the integral iteration to an acceptable level while maintaining reasonable accuracy. Typically, $k = 5$.

To compute the probability of successful operation of the components (i.e., those determined to be "1"), SIGPI integrates

BEST COPY AVAILABLE

from a lower limit of zero to an upper limit of $\mu+k\sigma$.

The similarity between the probability calculation and the complementary error function is used to calculate the probability of occurrence of a single component i which is included in the dependent component list. The probability P_i of component i is:

$$P_i = \frac{1}{2}\ \mathrm{ERFC}\left(\frac{1}{2}\ \frac{\mu_i}{\sigma_i}\right)$$

where ERFC is the complementary error function, μ_i' is the mean value of component i, σ_i ($\sqrt{V_{ii}}$) is the standard deviation, and V is the covariance matrix. This calculation is performed for each dependent component and will overwrite any probability value entered for that component in the independent component probability table.

If partitioning has taken place and there are dependencies among the upper partition (U) and its offspring partitions (O), the probability of a simple set in the offspring partition is

$$P(O|U) = \frac{P(O \text{ and } U)}{P(U)}\ ,$$

the conditional probability of the set in O given the set in U.

The multidimensional integral routine can handle up to 10 dimensions, or 10 mutually dependent components within a simple set. If more than 10 components are involved, upper and lower bounds for the subset are kept, but no additional decompositions can be made involving that mutually dependent data. Decomposition can continue using other data within the subset, if available.

Naturally occurring groups of data within the covariance matrix are utilized to allow independent treatment of groups of dependent data. The covariance matrix shown in Figure 3 contains two groups of mutually dependent data which are independent from each other.

When we can force the algebraic blocks described in the previous section to align with the covariance groups, we can take advantage of the partitioning algorithm. When using dependent data, conditioning the data to minimize the number of mutually dependent components can greatly enhance the overall accuracy and speed of the calculation by decreasing the size of the groups.

BEST COPY AVAILABLE

$$V = \begin{array}{c|ccccc} & C_1 & C_2 & C_3 & C_4 & C_5 \\ \hline C_1 & .25 & .11 & .02 & 0 & 0 \\ C_2 & .11 & .34 & .01 & 0 & 0 \\ C_3 & .02 & .01 & .15 & 0 & 0 \\ C_4 & 0 & 0 & 0 & .23 & .12 \\ C_5 & 0 & 0 & 0 & .12 & .15 \end{array}$$

Figure 3. Covariance matrix V in groups.

BEST COPY AVAILABLE

3. SIGPI USAGE

This section describes how to use SIGPI. The directions for operating SIGPI are given and the formats of the data files are defined. SIGPI reads two or three input files: the file which contains the disjoint normal sets and either a file which contains the conditionally independent probability data, or a file which contains the dependent probability data, or both. [See Figure 4.] Optionally SIGPI can write the resultant upper and lower bounds of the probability on the user's terminal or in an output file. SIGPI uses overflow files as needed for large data sets; these files should all be destroyed when the program finishes running. Additional input to the code can control the desired number of significant digits of agreement between the upper and lower bounds, and the number of conditional data sets to be used.

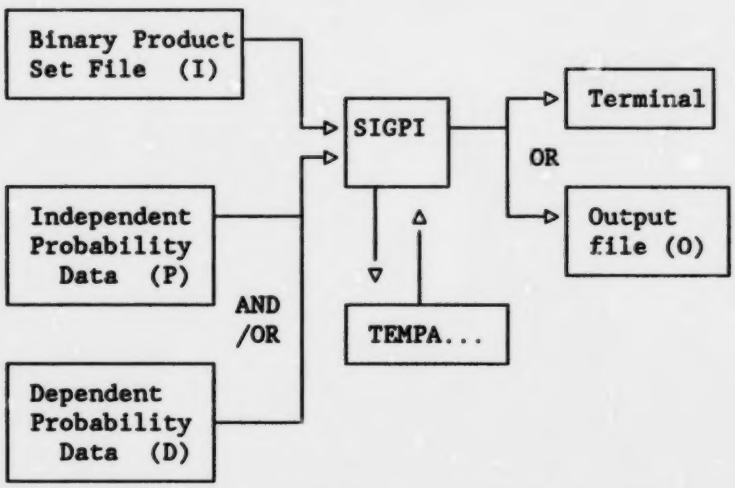


Figure 4. Diagram of SIGPI file usage.

3.1. Running the SIGPI program

SIGPI first types a prompt ">" and waits for input of a line of input describing files and options. Information about data files and options are input to the SIGPI program in the form

I-ifile [P-pfile] [D-dfile] [O-ofile] [ACC-n] [CON-k]

where capital letters represent required form, [] indicates optional input, and

ifile is the binary product set input file name.

BEST COPY AVAILABLE

SIGPI - Usage

pfile is the conditionally independent probability data file name.

dfile is the dependent probability data file name.

ofile is the optional output file name. The default is the terminal.

ACC = n allows accuracy determination for the calculation, where n is a positive integer determining the number of significant digits desired. The default accuracy is upper bound - lower bound.

CON = k specifies the number of conditional probability data sets in pfile and dfile. The default is one.

If no dependent components are included, an alternate input format is allowed with positionally defined arguments:

ifile pfile [n [k]].

3.1.1. Input Data Set File

The cut set information is input to the SIGPI program in the form of an unformatted file, each record of which represents one simple set. Each record contains the number of components contained in the set followed by the list of components contained in that set. Noncoherent data are negative numbers. For example, a system which consisted of three sets and four components

	C_1	C_2	C_3	C_4
S_1	0	0	0	-
S_2	-	0	-	0
S_3	0	-	-	1

could be represented by the following file:

Record	Data
1	3,1,2,3
2	2,2,4
3	2,1,-4

BEST COPY AVAILABLE

3.1.2. Probability Data Files

Conditionally independent data are input separately from dependent data. Each of these files may contain more than one set of conditional values. The conditionally independent data are input as a probability value for each component. The dependent data are input as statistical mean and covariance data. If only one type of data is used, the other file type need not be included. Both of these files are formated text input. Most FORTRAN's allow comma delimited numberic input fields. The formats used are included for use on systems which do not. For example, for most modern FORTRAN's a list of independent probability values could be input as:

```
1,8.3e-3
2,1.5e-4
3,.2
```

3.1.2.1. Conditionally Independent Data

The conditionally independent probability data file is a text file. It contains one complete set of probability values for each conditional probability desired. Each set contains one record for each probability value. The format is:

(I8,E20.5) - An eight digit component number followed by its probability in E or F format.

Each data set ends with a single record containing "*****" in the first 5 character positions.

To input two sets of conditional probability data sets for the example given in the previous section, the probability file could contain:

```
       1       .1
       2       .2
       3       .3
       4       .4
*****
       1       2.5E-3
       2       6.E-5
       3       1.2E-4
       4       4.3E-6
*****
```


BEST COPY AVAILABLE

3.1.2.2. Dependent Data

The dependent data file is also a text file. It may also contain more than one data set if conditioning has been applied to the data and the dependent data change for those conditions. Only the data that change in some aspect must be reentered. The format of each set is:

first record:
(I5) - Count of components in following data set (n).

n records of component data:
(I8,E20.10) - $Component_i$, $mean_i$

n*(n/4) records of the covariance matrix:
(4E20.10) - $Covariance_{j1,j2}$.

To input two sets of conditionally dependent data for the example given in the previous section, with components 1 and 2 independent of each other but both dependent on component 3 in the first set and components 1 through 3 dependent in the second set, the dependent probability data file would contain:

```
3
   1 3.2068711023E+00
   2 2.5000123778E+00
   3 2.8999976532E+00
3.2000156789E-01 0.0              1.7765900011E-01
0.0              2.8999211123E-01 1.0056700000E-01
1.7765900011E-01 1.0056700000E-01 2.5655434123E-01
3
   1 3.2068711023E+00
   2 2.5000123778E+00
   3 2.8999976532E+00
3.2000156789E-01 2.1111356745E-02 1.7765900011E-01
2.1111356745E-02 2.8999211123E-01 1.0056700000E-01
1.7765900011E-01 1.0056700000E-01 2.5655434123E-01.
```

Notice that no data is included for component 4 which is always independent of components 1 through 3 and whose probability value could be input in the independent data file.

EST COPY AVAILABLE

3.1.3. Output Data

Output from SIGPI is an upper and lower bound for each conditional data set. If an output file is selected, it is written in text mode. If not, output is to the terminal or controller. The format of the output is

```
('              Upper Bound                    Lower Bound')
(2E25.n),
```

where *n* is greater than or equal to the number of places of accuracy requested (up to the limit of the computer).

3.1.4. Accuracy Determination

The degree of accuracy desired may be specified on the parameter line. The accuracy is specified by the number of decimal places desired. This number is used as an epsilon value such that

$$\frac{P(\text{upper bound})-P(\text{lower bound})}{P(\text{upper bound})} \leq 10^{-n}.$$

The calculation will never go beyond 16 digits of agreement, and at that point the

$$P(\text{upper bound}) = P(\text{lower bound}).$$

3.1.5. Conditioning Control

If more than one data set is to be processed, the number of conditional data sets should be given on the parameter line in the form CON-k, where 'k' is the number of sets included. Each conditional set is delimited as defined for the probability input data files. If only one type of data changes because of conditioning, the other type need not be reentered; previously defined data will be retained.

BEST COPY AVAILABLE

4. PROGRAM IMPLEMENTATION

4.1. Program Structure

The SIGPI program is structured to make use of overlay capabilities and memory expansion techniques if they are available. The amount of computer memory to be used can be calculated based on the size of parameters desired--the number of components per set and the number of simple sets. The memory available must be able to hold the simple sets and sufficient working storage to hold several intermediate sets of subset parameters. For large problems, this memory allocation will not be sufficient to do the entire calculation. Excess data is stored in a disk file. In a non-time-sharing environment, the optimum memory allocation would be to use all that is available.

The control structure of the program is in several sections, overlayed and under the control of the main program. A general flow overlay structure diagram is included in Figure 4.

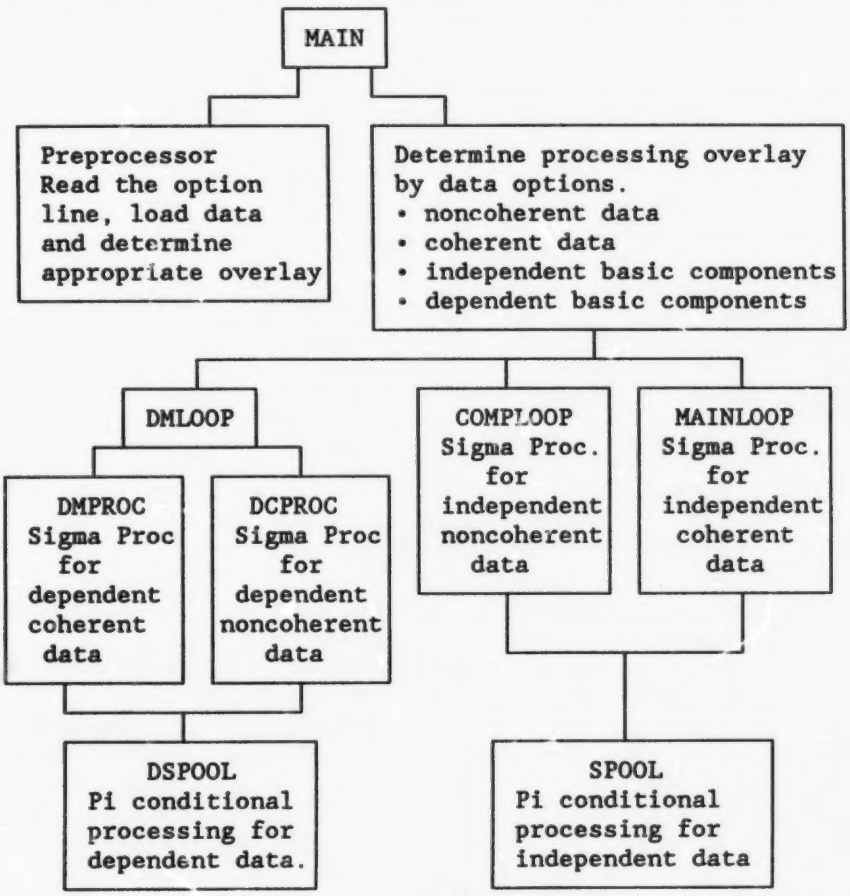


Figure 4. Flow diagram of SIGPI overlay structure.

BEST COPY AVAILABLE

4.1.1. The Control Program

The control program calls the preprocessor and then loads one of the main processor overlays depending on the options defined to the preprocessor. The main processor overlay calculates the cut set probabilities for all conditions given, calling in the spooling overlay if more than one condition is requested, and then returns to the main program which prints out a time usage table at the end of the job.

4.1.2. The Preprocessor

The preprocessor reads the option line for file names and options chosen. The cut set data and the first set of probability data for each basic component is loaded into memory.

4.1.3. The Main Processor

One of the processor overlays is chosen to be loaded depending of whether the data is coherent or noncoherent, and whether the basic components are independent or dependent. All of these overlays implement the $\Sigma\Pi$ algorithm. Figure 5 is a general structure diagram showing the major routines of the main processor overlay. The basic modules of all of the optional overlays are similar although details of the implementation change with the particular overlay. Appendix A gives the detailed overlay structure showing shared subroutines.

BEST COPY AVAILABLE

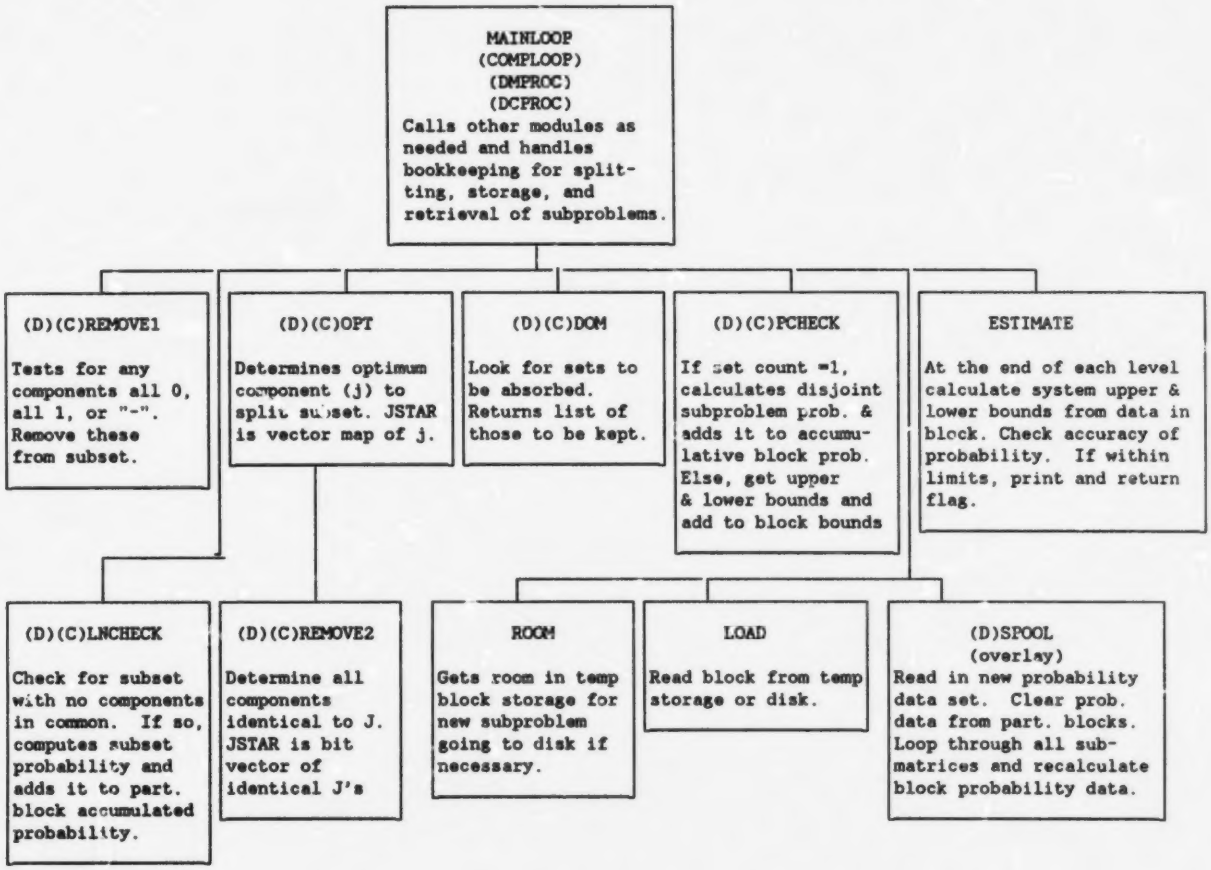

MAINLOOP
(COMPLOOP)
(DMPROC)
(DCPROC)
Calls other modules as needed and handles bookkeeping for splitting, storage, and retrieval of subproblems.
(D)(C)REMOVE1
Tests for any components all 0, all 1, or "-". Remove these from subset.
(D)(C)OPT
Determines optimum component (j) to split subset. JSTAR is vector map of j.
(D)(C)DOM
Look for sets to be absorbed. Returns list of those to be kept.
(D)(C)PCHECK
If set count =1, calculates disjoint subproblem prob. & adds it to accumulative block prob. Else, get upper & lower bounds and add to block bounds
ESTIMATE
At the end of each level calculate system upper & lower bounds from data in block. Check accuracy of probability. If within limits, print and return flag.
(D)(C)LNCHECK
Check for subset with no components in common. If so, computes subset probability and adds it to part. block accumulated probability.
(D)(C)REMOVE2
Determine all components identical to J. JSTAR is bit vector of identical J's
ROOM
Gets room in temp block storage for new subproblem going to disk if necessary.
LOAD
Read block from temp storage or disk.
(D)SPOOL
(overlay)
Read in new probability data set. Clear prob. data from part. blocks. Loop through all submatrices and recalculate block probability data.

Figure 5. Structure diagram of major SIGPI subroutines.

4.2. Data Management

This section deals with memory variable blocks and temporary disk usage. Data storage that remains static for each system analyzed by SIGPI is defined first and contains the basic component arrays for names, mapping, and probability values, and the matrix of cut sets. Two sets of data blocks are defined to describe the subproblem currently being disjointed. Partitioning information is kept in a tree-structured array set. Dynamically controlled memory contains the stored subproblems. In order to conserve space, the matrix is not recopied into each subproblem. Rather, tables are maintained that contain all of the information about each subproblem.

BEST COPY AVAILABLE

4.2.1. Basic Component Data

The basic component data is contained in three arrays.

NAMES-The names of the components as defined in the input file.

MAP- Table that correlates the component names with the bit positions used to store the cut set information. There is one entry for every component used in the cut set, in the order that the components are internally represented. The table contains the index to the name table.

PTAB- Table of probability values for each component used in the cut sets. One conditional set is held in memory at a time.

4.2.2. The Cut Set Matrix

Each simple set in the matrix is stored as a bit vector in array ISSET, each bit representing one component; a 1 in the bit position represents a component with a "0" value in the simple set.

If noncoherent data is present in the input, a second array, ICSET, contains the bit vectors for the complemented components for each simple set. For each simple set, therefore, a bit position that contains 0 in both ISSET and ICSET arrays would be a "don't care" component. The advantage of representing the simple sets in this manner is that it allows manipulation by Boolean arithmetic in a fast and straightforward manner.

4.2.3. Partitioning Block Tree

Every subproblem belongs to a partitioned block. If a subproblem has never been part of a problem that has been partitioned, it points to block zero. As a subproblem is partitioned into two or more blocks, each partitioned subproblem and all of its offspring subproblems point to a unique entry in the PBLK array. For independent basic components, each entry in the PBLK array contains three values:

1) The accumulated probability of the disjointed subproblems which belong to this block.
2) The accumulated upper bounds of subproblems which belong to this block but which are not yet disjoint.
3) The accumulated lower bounds of subproblems which belong to this block but which are not yet disjoint.

BEST COPY AVAILABLE

If dependent basic components are present, there are six values:

1) The upper estimate of the probability of the disjointed subproblems which belong to this block.
2) The lower estimate of the probability of the disjointed subproblems which belong to this block.
3) The accumulated upper bounds of subproblems which belong to this block but which are not yet disjoint.
4) The accumulated lower bounds of subproblems which belong to this block but which are not yet disjoint.
5) The upper bounds for subproblems which cannot be calculated because of dependency problems.
6) The lower bounds for subproblems which cannot be calculated because of dependency problems.

The upper and lower estimates are necessary because of the nature of the integral used to calculate the probability of mutually dependent basic components. This calculation attempts to reach a desired accuracy within memory and time constraints. As the number of mutually dependent components increases, the accuracy actually received decreases. A maximum of ten mutually dependent basic components can be handled. When the disjointing process generates sets with more than ten mutually dependent basic components, the process can go no further. Other non-dependent basic components are used to process these subproblems as far as possible and then the upper and lower bounds of these sets are maintained in positions 5 and 6 of the PBLK array. For this reason, data sets that contain a large number of mutually dependent basic components sometimes cannot be calculated to the accuracy requested.

Each PBLK group also has associated pointers in the three arrays: ULINK, LLINK, and JLINK.

LLINK — Points to the previous or sibling block of this partitioning. A zero in this field means that it is the first block of the partition. At the end of each level, the system probability is calculated going from the last to the first entry in PBLK, so a zero signals the end of processing for this block group.

ULINK — Points to the upper block or parent block of this partition. All sibling blocks point to the same parent block.

JLINK — Points to the JBLK entry for this group. All sibling blocks point to the same JBLK entry.

JBLK is an array of bit vectors that contains the list of components that had already been factored from a subproblem by the disjointing process before it was partitioned. JBLK has an associated JBLKCTR array which contains the number of vectors in the JBLK entry for this partition. All JBLK entries contain at

BEST COPY AVAILABLE

least two vectors, the JA0 and JA1 vectors, for independent basic components which have been disjointed. Additionally, for systems with dependent basic components, a JA0 and JA1 vector is kept for each group of mutually dependent basic components. Any vectors defined by the JQ array are also kept here. (See Figure 6 for a definition of JA0, JA1, and JQ.) If dependent basic components are present, an additional array is maintained, JBLKPS, which contains the upper and lower probability estimate for each vector in JBLK.

These arrays are allotted 200 entries each. If they overflow that amount, no additional partitioning takes place.

4.2.4. Subproblem Temporary Storage Area

So that the simple sets will not be duplicated as each problem is split into two subproblems during the disjointing process, each division is represented by a set of control words and arrays. As each problem is picked up for processing, it is stored in the CUR data block structure. As the problem is being split along the optimum component J, the table information for the part of the subproblem in which J is being assigned a value of "1" is stored in the data block NEW. The remaining part of the subproblem in which J is "0" remains in CUR. NEW and CUR are then stored at the end of the temporary storage area. The original data set which was contained in CUR is discarded and CUR is loaded with the next subproblem in the list, each problem being handled on a first-in-first-out basis.

Figure 6 contains the data structures stored in temporary memory. Subproblem temporary storage is handled as a large circular buffer, guaranteeing that all subsets within a level are processed before continuing with the next level.

Figure 7 contains a diagram of memory usage. If the available space in memory fills up, temporary disk files are used to hold the excess subproblems. The block of temporary memory is currently allocated 33280 words of memory. Figure 8 contains a diagram of disk usage.

BEST COPY AVAILABLE

Name in CUR	Name in New	See Notes	Size in Block	Description
SETCT	RSETCT		1	Total word count in the subproblem control block.
LEVEL			1	Level this subproblem belongs to.
CARRY	NCARRY		1	Processing is completed for this subproblem. Data is preserved if another conditional data set will be used.
ICT	ICNEW		1	Number of simple sets in this subproblem.
NJS	NJSR	1	1	Number of vectors needed to describe each mutually dependent set of disjointed basic components (see JSV vector).
NJQ	NJQR		1	Number of JQ vectors in this subproblem.
NPB			1	Pointer to PBLK.
PARTP	PARTPNEW	2	1	Partial probability of the subproblem,i.e. the probability of the factored components. The product of the probability values of the components in JA0 times the product of (1.-probability values) of the components in JA1 times (1.-the product of the probability values of the components in each JQ).
PLOW	PLOWNEW	1,2	1	The lower bound of the subproblem.
JALPHA	JANEW	3	NCTRL	Bit vector representing the components not yet determined to be "0" or "1".
JTOT	JTOTNEW	1	NCTRL	Bit vector representing all components that have been processed.
JA0	JA0NEW	4	NCTRL	Bit vector representing the components determined to be "0".
JA1	JA1NEW	4	NCTRL	Bit vector representing the components determined to be "1".
PJS	PJSN	1,2	4*NJS	The upper and lower estimates for the "0" and "1" vectors described in JSV.
JSV	JSVN	1	NJS*2 *NCTRL	Set of vectors representing the mutually dependent components determined to be "0" or "1". The first two always represent the independent basic components. These are followed by sets of vectors representing mutually dependent groups of components determined to be "0" or "1". The last ones are those described by the NJQ counter.
JQ	JQNEW		NJQ *NCTRL	A bit vector defined each time JSTAR contains more than one optimum J. Each one contains the JSTAR components on the NEW (or "right-hand") split of the problem.
IALPHA	IANEW		ICT	An array containing the list of indices of the simple sets in this subproblem. Defines which bit vectors in ISSET and ICSET are included.
LMA	LMANEW		ICT	An array of counters for each simple set defined by IALPHA. The array contains the number of "don't care" components remaining in the simple set.
LNA	LNANEW		NCT	An array of counters for each component. The array contains the number of simple sets where this component has a "0" value.
LNA1	LNA1NEW	5	NCT	An array of counters for each component . The array contains the number of simple sets where this component has a "1" value.

Figure 6. Subproblem control blocks.

NOTES:
1. Dependent option only.
2. Real numbers, may take multiple words of storage in smaller computers.
3. NCTRL is the number of words required to hold a bit vector. NCTRL = (NCT-1)/WORDSIZE+1, NCT=total number of basic components in the system.
4. Independent option only.
5. Noncoherent option only.

BEST COPY AVAILABLE

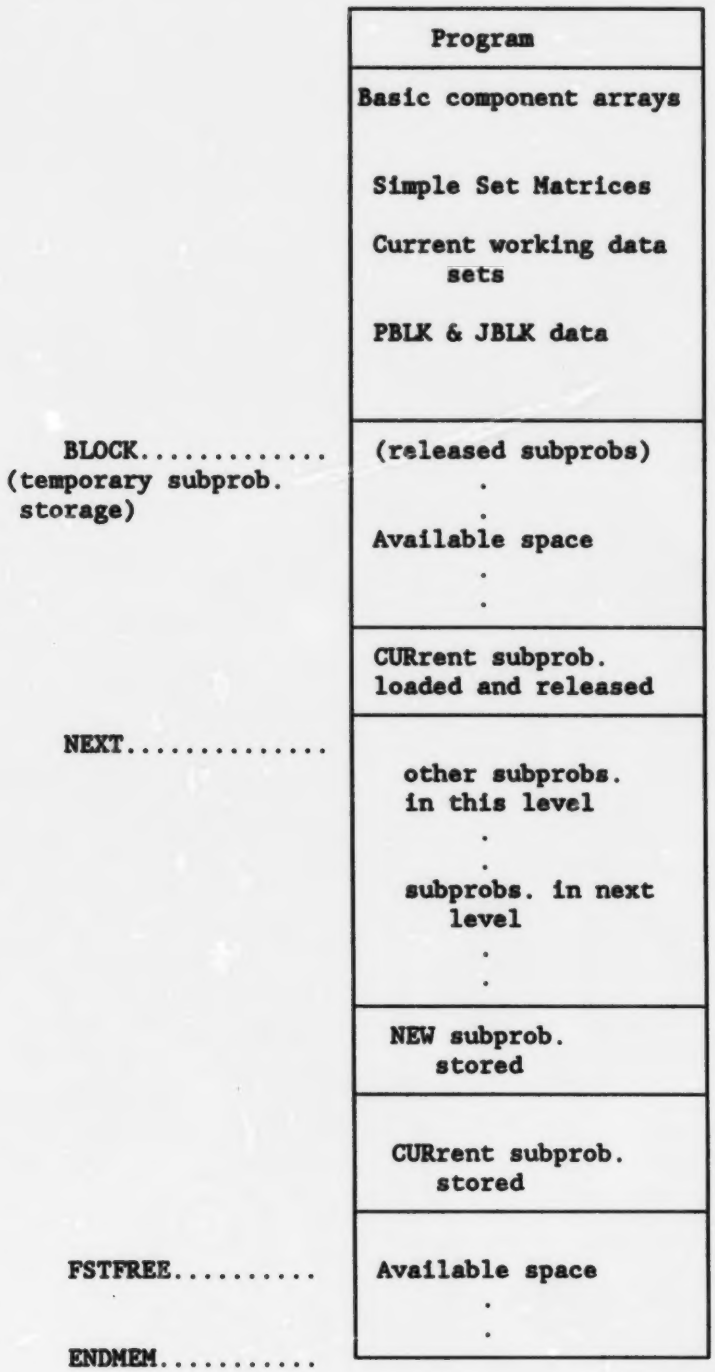


Figure 7. Diagram of memory usage.

BEST COPY AVAILABLE

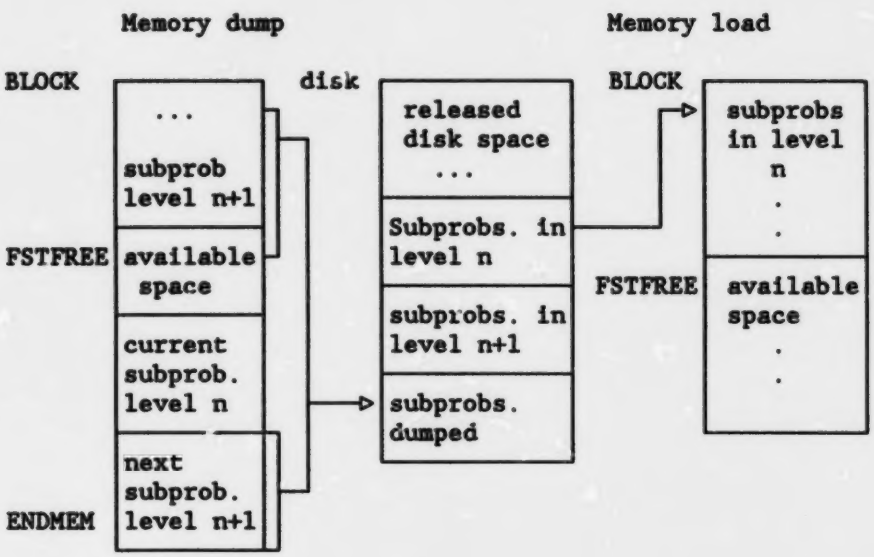


Figure 8. Diagram of memory/disk transfer.

4.3. Implementation of the ΣΠ Algorithm

This section presents an overall description of the main processor algorithm. All variable and subroutine names used in the program are written in capitals. Subroutine names that have a (C) in front of them have a separate version for noncoherent systems. Those with a (D) have a separate version for dependent basic components. A shorthand notation of the FORTRAN implementation is included where appropriate.

We assume that the MCT failure or cut sets for a system with NCT components are available. We further assume that a single probability of failure value (PTAB) is available for each independent basic component, and that mean and covariance data are available for each mutually dependent basic component. The probability calculation is included in each appropriate step.

BEST COPY AVAILABLE

4.3.1. Definition of the Σ Algorithm

Each subproblem in the system is processed in the following manner:

1. (D)(C)REMOVE1 - For independent basic components, remove any components where all sets have either a "0", a "1", or a "don't care". The "don't care" component is irrelevant to the subproblem. The "0" condition or the "1" condition is defined for all sets in the subproblem and can be moved to the partially disjointed sets which are being defined. The probability of the removed component can be included in the partial probability product.

   For dependent basic components, D(C)REMOVE1 handles only the components in which all sets have a "don't care". A separate subroutine, D(C)REM0, handles the special case where all components have a "0" or a "1":

   ```
   if LNA(J)=0 and LNA1(J)=0 then JALPHA(J)=0

   if LNA(J)=ICT      then JA0(J)=1,
                           JALPHA(J)=0,
                           PARTP=PARTP*PTAB(J)[1#]
   if LNA1(J)=ICT     then JA1(J)=1,
                           JALPHA(J)=0,
                           PARTP=PARTP*(1.-PTAB(J)).
   ```

2. (D)(C)LNCHECK - Determine if there are any components in common in the simple sets within the subproblem. Where there are no components in common (each component has a "non-don't care" value in only one set), we may determine the probability of the subproblem without further disjointing. For each component determine the number of simple sets where that component has either a "0" value (LNA) or a "1" value (LNA1). The partial probability of the subproblem can be calculated and added to the accumulative probability if the number of simple sets listed in LNA and LNA1 is always less than or equal to 1, and the remaining components to be evaluated in each set are mutually independent from the components already determined and the components of other remaining sets. For each *j* component in each simple set *i*:

   ```
   P(simple set i)=Product of PTAB(j),[2#]  if ISSET(j,i)=1
                  * Product of (1.-PTAB(j)), if ICSET(j,i)=1
   ```

1# In the case for dependent basic components, the joint mutual probability is calculated.

2# In the case of dependent basic events, the joint probability of the coordinates in each set is calculated.

BEST COPY AVAILABLE

```
P(subproblem)= PARTP*
          (1-[Product of (1- P(simple set i)),
             for i=1,ICT])

Accumulate block probability
          (PBLK(1,n) = PBLK(1,n) + P(subproblem)
CARRY = TRUE
GO TO step 10.
```

3. (D)(C)BLKTEST - Determine if the subproblem can be partitioned (see section 4.4.2). If so, store all new partitions except the last as new problems. Proceed with disjointing the last partitioned block.

4. (D)(C)OPT - Determine the optimum component, JSTAR, along which to disjoint the simple sets into subproblems (see section 4.4.3).

5. (D)(C)REMOVE2 - See if any more components behave identically to the optimum component in all simple sets. If so, JSTAR is bit vector of all identical components.

6. Replace the parent problem with two new subproblems. The first, "right-hand", problem consists of all simple sets of the parent problem with component JSTAR equal to "1" (BLIST). The second, "left-hand", problem consists of all those simple sets from the parent problem that have a "0" in component JSTAR (ALIST). Those simple sets (CLIST) which have a "don't care" in component JSTAR go to both new problems, with the JSTAR component set equal to "1" for the "right-hand" problem, and set equal to "0" for the "left-hand" problem. For the overlays for dependent basic components, the subroutines DANAL and JMIX separate the components so determined into mutually dependent vectors.

```
Left-hand:    JA0(JSTAR)=1,
              JALPHA(JSTAR)=0,
              PARTP=PARTP* P(JSTAR)

Right-hand:   JANEW(JSTAR)=0,
              PARTPNEW=PARTPNEW*(1-P(JSTAR))
    if JSTAR has single value
              JA1NEW(JSTAR)=1
    if JSTAR multiple values
              DQN=DQN+1
              JQNEW(DQN)=JSTAR
```

7. (D)(C)DOM - For both new subproblems, remove every simple set which is contained in any other. The simple sets which might be contained will be those in CLIST. The ones in CLIST which cannot be contained will be returned in DLIST.

BEST COPY AVAILABLE

8. Adjust the values of the subproblem control arrays for the left (CUR) subproblem and for the right (NEW) subproblem. See Figure 9 for the list of the subproblem control block contents.

9. (C)PCHECK - If either new problem consists of a simple set, this simple set is a disjoint product of the procedure. Add the simple set probability (PARTP multiplied by the joint probability of any remaining "non-don't care" components in ISSET or ICSET) to the accumulative block probability (PBLK(1,n)). (For dependent basic components, this procedure is handled in the main processing program rather than in a separate subroutine.) Set CARRY-TRUE.

   If the new problem consists of more than one simple set, calculate the upper and lower bounds for the subproblem. For independent basic components, the upper bound is the sum of the simple set probabilities and is added to the accumulative block upper bound (PBLK(2,n)). The lower bound is the maximum of the simple set probabilities and is added to the accumulative block lower bound (PBLK(3,n)). For dependent basic components, the upper bound is the partial probability of the components already disjointed, PARTP. The lower bound is PARTP multiplied by the joint probability of the largest single set, using the PLO and SMIX routines to determine and evaluate that set.

10. If either new problem consists of more than one simple set, it becomes a new subproblem for the next level of processing. If not, set CARRY-TRUE; and if the number of conditional probability data sets (NCON) is greater than 1, then keep the marked set for subsequent conditional data set processing.

11. If there are more subproblems at this level, take the next subproblem from the problem list and return to step 1.

12. (D)ESTIMATE - If there are no more problems at this level, calculate the system bounds. If the estimate is not yet accurate to the decimal place requested, process the next level by returning to step 1.

13. (D)SPOOL - If the estimate is within the desired accuracy and there are more conditional data sets, then load the new values into the probability table (PTAB), and recalculate the subproblem partial probabilities (PARTP) and the partitioned block accumulated probabilities and bounds (PBLK). When using dependent basic components, it is also necessary to 1) realign the vectors of mutually dependent components if the dependencies have changed and 2) reevaluate the probabilities of the JBLK entries. Recalculate the system estimate. If the estimate is not within the desired accuracy, return to step 1 to continue processing with the new probability data.

BEST COPY AVAILABLE

4.3.2. The Partitioning Algorithm - (D)(C)BLKTEST

This subroutine tests the subproblem for partitioning. If the conditions of commonality and independence are met, the subproblem is split into two sections and the tree structure defining this partition is generated. Steps 1) through 4) test for these conditions, and steps 5) through 7) do the partitioning.

1) If there are at least five simple sets in the subproblem and if no simple set consists of only one component which has a "non-don't care" value, then test for partitions in the subproblem. Make a list of all components in the first set that have either a "0" or a "1" value.

2) For each additional simple set in which a component in the list contains either a "0" or a "1" value, add to the list all components in that set that contain either a "0" or a "1" value.

3) If any components have been added to the list, repeat step 2.

4) If the list does not contain all of the components in the subproblem, and none of the components contained in the list is mutually dependent with a component not in the list but not yet processed, partitioning is possible. If not, and we have partitioned any blocks during this entry, process the last one; otherwise, return to the disjointing process.

5) In JBLK, save a list of all components which have been factored from the subproblem by previous disjointing operations. For independent basic components, this is the contents of the JA0, JA1, and all JQ vectors. For dependent basic components, this is the contents of the JSV array. Generate a tree linkage (PBLK) with pointers to the JBLK entry (JLINK), to the block to which this subproblem belonged before this partitioning operation (ULINK), and between the newly partitioned blocks (LLINK).

6) Generate a NEW subproblem which consists of the components in the list defined in steps 1 through 3 above, and all of the simple sets in which any component in the list contains either a "0" or a "1" value. Use (D)(C)PCHECK to determine an upper and lower bound, and store this subproblem at the end of the problem list.

7) Generate another subproblem, CUR, which consists of all remaining components and simple sets. Return to step 1 to see if further partitioning is possible.

BEST COPY AVAILABLE

4.3.3. Optimum Component for Disjointing - (D)(C)OPT

This subroutine examines the subproblem to determine the optimum component for disjointing. The component is chosen to minimize the size of the two subproblems being generated while trying to maintain dependency groupings. If during any step of this procedure, only one J satisfies the conditions, no further steps are applied.

1) Look for a component which contains either a "0" or a "1" value in every simple set (LNA(J) + LNA1(J) = ICT). If present, this is the optimum component J. Use this option for noncoherent systems only.

2) Determine the list of components that contain a "0" or a "1" value in any simple set which has a maximum number of "don't cares",

   TEMP(j) = ISSET(j,I) OR ICSET(j,I), I=MAX (LMA),
   for all j's in JALPHA.

3) From the components chosen in step 2, determine the components which have a "0" or a "1" value in the maximum number of simple sets.

   TEMP(j) = TEMP(j) AND TEMP(k), k=MAX (LNA+LNA1)
   for all j's chosen in 2).

4) For noncoherent systems and using the components chosen in step 3, determine the components in which the number of simple sets where that component has a "0" value most equals the number of simple sets where that component has a "1" value.

   TEMP(j) = TEMP(j) AND TEMP(k), k=MIN(|LNA-LNA1|)
   for all j's chosen in 3).

5) From the components chosen in step 4, determine the maximum number of "don't care" components in a simple set where the chosen component has a "0" or a "1" value.

   $MAXLM_j$ = MAX (LMA(i)), if ISSET(j,i)=1 or
   ICSET(j,i)=1
   for all j's chosen in 4).

6) From the components chosen in step 4, determine the minimum number of "don't care" components in a simple set where the chosen component has a "0" or a "1" value.

   $MINLM_j$ = MIN (LMA(i)), if ISSET(j,i)=1 or
   ICSET(j,i)=1.
   for all j's chosen in 4).

BEST COPY AVAILABLE

7) For independent basic components, the optimum component J is the one with the least product of the results of step 5 weighted with a fudge factor and the results of step 6. If more than one component satisfies these tests, the first or leftmost one is chosen.

$$J = \text{MIN}(\text{MAXLM}_j * \text{MAXLN} + \text{MINLM}_j),$$
for all j's chosen in 4).

For dependent basic components the number of other components each J is dependent on is taken into account. In this case the equation is:

$$J = \text{MAX}(\text{DEP}(j) - \text{MAXLM}_j * \text{MAXLN} + \text{MINLM}_j$$
for all j's chosen in 4).

4.3.4. The System Probability ESTIMATE (DEST)

At the end of each level of processing, estimate the system probability by calculating the upper and lower bounds of the system. This calculation uses the PBLK partitioning block control structure. All of the probability calculations and bounds that have been generated at each level by the partitioning and disjointing algorithms have been stored in the PBLK array. Each PBLK entry contains three probability values (six for dependent basic components): the accumulated disjointed subproblem probabilities, the accumulated upper bounds and accumulated lower bounds for subproblems which are not yet disjoint, and the upper and lower bounds for subproblems which cannot be processed further because of dependency problems. The probability estimate proceeds from the last group of entries in the PBLK array, those most recently partitioned, to the first. The algorithm used for calculating the system estimate at the end of each level is:

1) Pick up the last (nth) block of a partitioned sibling group. The link to another block of this group is in LLINK(n). The link to the JBLK entry is in JLINK(n). For all blocks (m,...,n) in this group, calculate an upper and lower bound. Add the accumulated disjoint subproblem probability respectively to the accumulated subproblem upper bounds to get the group upper bound, PTU, and to the accumulated subproblem lower bounds to get the group lower bound, PTL.

BEST COPY AVAILABLE

For independent basic components the equations are:

```
PTU = 1. - [ (1.-(PBLK(1,m)+PBLK(2,m)))
           *(1.-(PBLK(1,m+1)+PBLK(2,m+1)
           * ...
           *(1.-(PBLK(1,n)+(PBLK(2,n)))])

PTL = 1. - [ (1.-(PBLK(1,m)+PBLK(3,m)))
           *(1.-(PBLK(1,m+1)+PBLK(3,m+1)
           * ...
           *(1.-(PBLK(1,n)+(PBLK(3,n)))])
```

For dependent basic components the equations are:

```
PTU =  1. - [ (1.-(PBLK(1,m)+PBLK(3,m)+PBLK(5,m)))
           *(1.-(PBLK(1,m+1)+PBLK(3,m+1)+PBLK(5,m+1)))
           *...
           *(1.-PBLK(1,n)+PBLK(3,n)+PBLK(5,n)))]

PTL = 1. - [ (1.-(PBLK(2,m)+PBLK(4,m)+PBLK(6,m)))
           *(1.-(PBLK(2,m+1)+PBLK(4,m+1)+PBLK(6,m+1)))
           *...
           *(1.-PBLK(2,n)+PBLK(4,n)+PBLK(4,n)))]
```

Clear the accumulated subproblem upper and lower bounds (but not the accumulated disjoint subproblem probabilities) in each block to prepare for the next level.

2) When the bounds of all of the linked partitions have been calculated, multiply them by the probability values of the components that had been factored before this subproblem was partitioned. These components are stored in the JBLK array. The JBLKCTR array contains the number of vectors in this entry. For independent basic components, the JBLK contains the bit vectors that correspond to the original JA0 and JA1 vectors and the JQ vectors. The probability of the components in the JA0 vector is the product of the probabilities of those components.

$P(JA0) = PTAB(j_1) * PTAB(j_2) * \ldots * PTAB(j_n)$,
for all "non-don't care" j's in JA0.

The probability of the components in the second (JA1) vector is the product of the probabilities of the complements of the components in the vector.

$P(JA1)=[1.-PTAB(j_1)]*[1.-PTAB(j_2)] *\ldots* [1.-PTAB(j_n)]$
for all "non-don't care" j's in JA1.

BEST COPY AVAILABLE

The probability of each of the JQ vectors is the complement of the product of the probabilities of those components.

$$P(JQ)=1.- [PTAB(j_1) * PTAB(j_2) * \ldots * PTAB(j_n)]$$
for all j's in JQ.

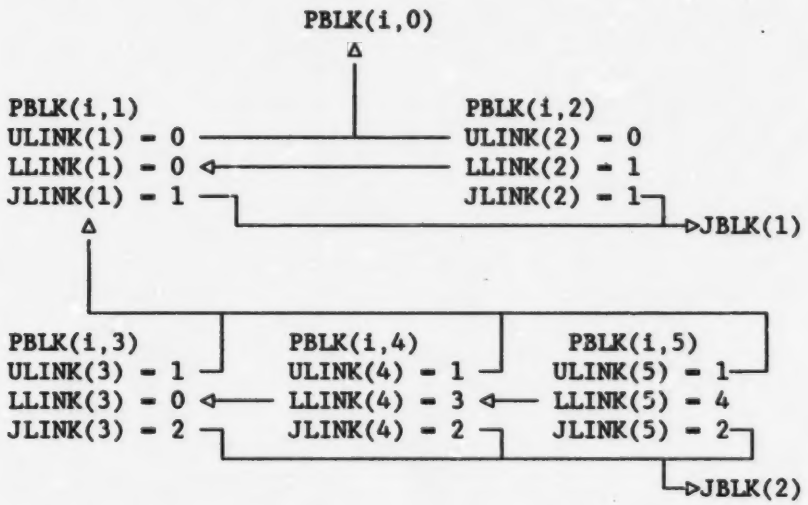
The upper and lower bounds are multiplied by the product of these probabilities.

$$PTU=PTU * P(JA0) * P(JA1) * P_1(JQ) * \ldots * P_n(JQ),$$
for each JQ vector.

$$PTL=PTL * P(JA0) * P(JA1) * P_1(JQ) * \ldots * P_n(JQ),$$
for each JQ vector.

For dependent basic components, the JBLK array contains the contents of the JSV array at the time of the partitioning. Additionally, there is a JBLKPS array which contains the upper and lower estimates of the probabilities of each of these vectors. These values are multiplied by the PTU and PTL values in a similar manner.

3) The link to the upper block from which this partitioning took place is in ULINK(m). The bounds of the linked group is added to the upper block upper and lower bounds entry. For example, assume that the original problem set, PBLK(i,0), was partitioned into two blocks, PBLK(i,1) and PBLK(i,2), and the first partitioned block was further partitioned into three more blocks, PBLK(i,3), PBLK(i,4), and PBLK(i,5). The linkage entries would be as follows:



BEST COPY AVAILABLE

After PTU and PTL were calculated for blocks 3 through 5, they would be added to the upper and lower bounds entries of block 1.

```
PBLK(2,1) = PBLK(2,1) + PTU
PBLK(3,1) = PBLK(3,1) + PTL   [3#]
```

PTU and PTL would then be calculated for blocks 1 and 2 and would be added to the upper and lower bounds entries for block 0.

4) If the last block index is greater than 1, go to step 1 to calculate the bounds of another linked group of partitioned blocks.

5) The entries for PBLK(i,0) are the bounds for the system as a whole after all of the linked subgroup bounds have been added to them. This first entry set contains the probability data for the subproblems that have never been partitioned, as well as the "glue" pieces we have just added from the partitioned blocks. The probability of the disjointed subproblems, PBLK(1,0), added to the upper and lower estimates, PBLK(2,0) and PBLK(3,0),[4#] gives the upper and lower bounds for the system. The system bounds are contained in variables UPPER and LOWER, and the bounds entries of PBLK are cleared for the next level.

6) If the difference between the upper and lower bound is less than the desired accuracy, print the bounds on the output device and return a TRUE value. Otherwise return a FALSE value.

4.4. Example of The Σ Algorithm

The example shown in Figure 9 shows the disjointing, partitioning, and bounds estimating for a sample system matrix using independent basic components. For a problem with dependent basic components, the disjointing and partitioning would be identical, but a different equation is used for the bounds estimation. The problem consists of components C1 through C9 in

---

[3#] For dependent basic components, the indices would be (3,1) and (4,1), respectively.

[4#] For dependent basic components, the upper and lower disjoint subproblem estimates in PBLK(1,0) and PBLK(2,0) are added to the upper and lower bounds estimates in PBLK(3,0) and PBLK(4,0) and to the upper and lower bounds of the unsolvable subproblems in PBLK(5,0) and PBLK(6,0).

BEST COPY AVAILABLE

simple sets S1 through S7. Each component has a probability value in the PTAB table that is represented in the figure by the notation PCn. Examination of the original system matrix shows that the matrix contains two blocks, one using components E2, E3, and E4, and one using components E5, E6, E7, E8, and E9, with component E1 tying the two blocks together. This matrix cannot be partitioned, but it can be disjointed.

Going to the OPT procedure described in section 4.4.3 to pick the optimum component for disjointing, we find that the simple sets S1 through S7 have sizes defined by the array LMA that consists of six "don't care" components each. Therefore, at the end of step 2, we have all of the components in our list. Step 3 picks component C1 as having the most "non-don't care" components. Since we have a single component, we skip the remaining tests. We also find no component identical to C1.

We evaluate our optimum component C1 and split the simple sets accordingly. C1 has a "0" value in sets S1 through S5, so these sets go only to the "left-hand" (CUR) subproblem. C1 has a "don't care" value in sets S6 and S7, so these sets go to both the "left-hand" and the "right-hand" (NEW) subproblems. We first define the NEW subproblem. The C1 bit set in the JANEW1 vector, shown in Figure 9 as (C1=1) to define components set in the JA1 vectors and as (C1=0) to define components set in the JA0 vectors. PARTPNEW is set to (1.-PC1), the partial probability value defined for the right hand subproblem. The upper bound for the subproblem is calculated by summing the probabilities of the simple sets and multiplying that by PARTPNEW. The lower bound is the maximum probability of the simple sets, multiplied by PARTPNEW. The bounds are placed in the upper and lower bound positions of PBLK(0), shown on the diagram as UB and LB respectively. The disjointed partial probability shown on the diagram as DP has a zero value.

All of the simple sets go to the "left-hand" (CUR) subproblem. The C1 bit is set in the JA0 vector, and PARTP is set to PC1. Similarly, the upper and lower bounds are calculated and added to the values in PBLK(0). They can be added because these submatrices are disjoint along the C1 component. Both subproblems are stored at the end of the subproblem list. Since there are no more subproblems at level one, we estimate the system probability.

Since no partitioning has been done and no completely disjointed subproblems have been processed, the estimation at this time is a comparison of the upper and lower bounds in PBLK(0). If the estimated upper and lower bounds are not sufficiently close at this time, we will proceed to Level 2.

The "left-hand" subproblem may be partitioned now. The first (A) block's JALPHA entry consists of components C2, C3, and C4 and IALPHA has simple sets S1, S2, and S3. This block would actually be attached to PBLK(1), but it is shown here attached to PBLK(2)

BEST COPY AVAILABLE

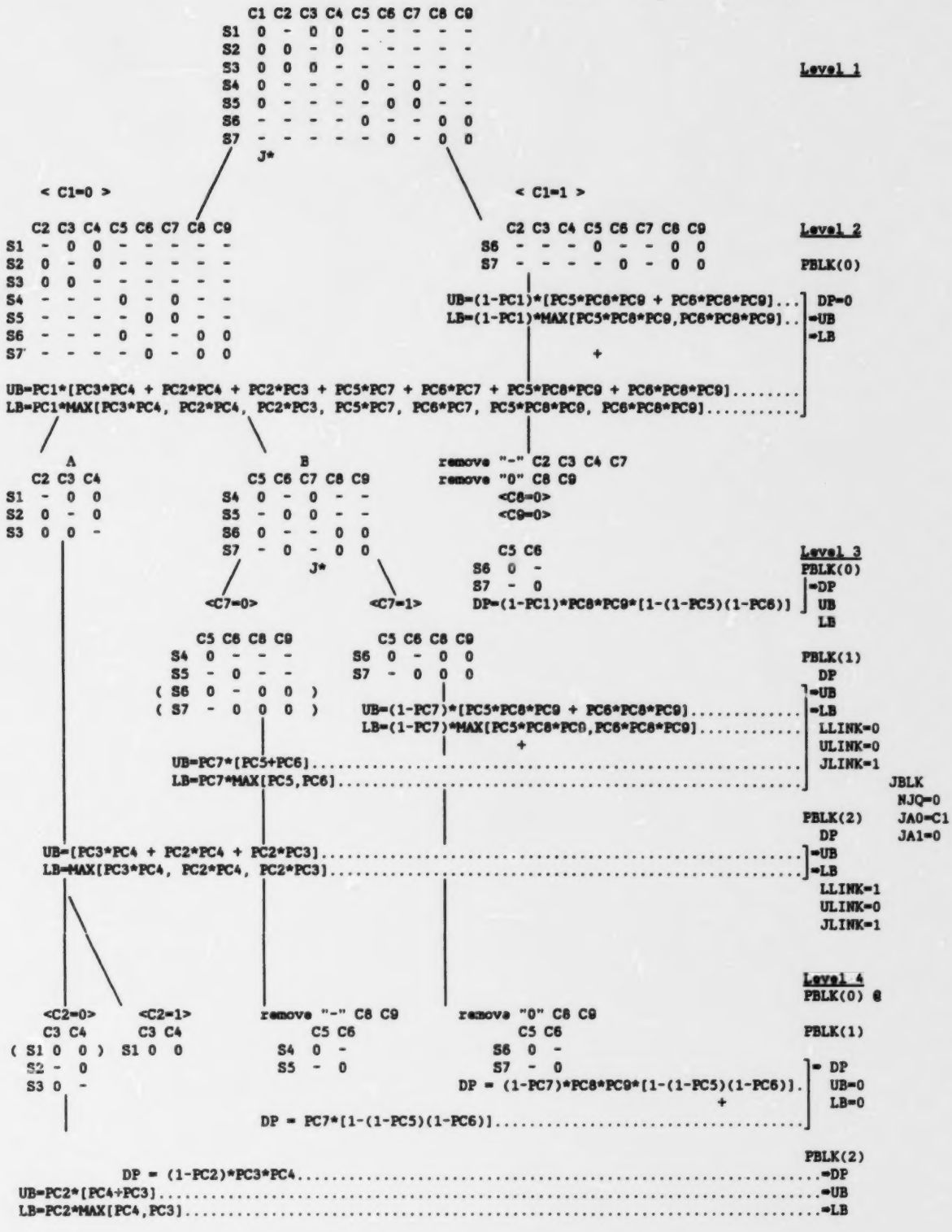


Figure 9. Example Calculation.

BEST COPY AVAILABLE

because it makes the diagram easier to read and PBLK(1) and PBLK(2) are linked to create equivalent entries. The PBLK entries are linked to a JBLK entry, and the contents of JA0 and JA1 are placed in the JBLK entry, there being no JQ vectors in the subproblem. JA0, JA1, and PARTP are cleared. The upper and lower bounds are calculated and added to the appropriate spots in its PBLK entry.

The second (B) block of the partitioning consists of components C5 through C9 and simple sets S4 through S7. Its PBLK entry is linked to the first partition and to PBLK(0) as its "upper" link. JA0, JA1, and PARTP are also cleared for this block. Since block B cannot be partitioned further, it proceeds to the disjointing algorithm.

Step 1 of the disjointing algorithm does not apply so we proceed to step 2. The size of the simple sets as defined in the LMA array gives us the components defined in simple sets S4 and S5, their having three "don't care" components each. Components C5, C6, and C7 each have a "0" value in two simple sets, so step 3 does not eliminate any of them. Step 4 does not apply to a coherent system. Step 5's search for a maximum LMA value in a simple set where the component has a "don't care" value, yields 3, 3, and 2 respectively. Step 6's search for a minimum LMA in a simple set where the component has a "0" value, yields 2, 2, and 3. Step 7 looks at the weighted product of these values, which in this case gives us W3*2, W3*2, and W2*3, and picks C7 as having the least product. C7 has no identical component, and it is placed in JSTAR.

The block is split into two subproblems similar to the Level 1 split. The NEW subproblem consists of components C5, C6, C8, and C9 and simple sets S6 and S7. The JA1NEW vector contains C7 and PARTPNEW contains (1-PC7). The remaining simple sets in CUR also consist of components C5, C6, C8, and C9. Two simple sets S6 and S7 can be absorbed or contained by sets S4 and S5. The upper and lower bounds of both these subproblems are calculated and summed in their PBLK entry.

There is one more subproblem on level two: the subproblem generated by the "right hand" split in level 1. Examination of it shows that components C2, C3, C4, and C7 have a "don't care" value in both of the simple sets, so they can be removed from the problem. Similarly C8 and C9 are "0" in both simple sets, so they can be removed to the JA0 vector and PC8 and PC9 multiplied into PARTP. We have components C5 and C6 remaining, and the condition of having no components in common between the two simple sets, S6 and S7. We can calculate the probability of this subproblem completely, and it is placed in the disjointed subproblem probability entry, DP in PBLK(1,0).

We are now ready to calculate the system estimate for level 2. The last PBLK entry is picked up as well as its sibling PBLK

BEST COPY AVAILABLE

entries. The upper and lower bounds for the combination are calculated (PTU and PTL),

```
PTU= 1.- (1.-DP(1)+UB(1)) * (1.-DP(2)+UB(2))
PTL= 1.- (1.-DP(1)+LB(1)) * (1.-DP(2)+LB(2)).
```

These values are multiplied by the probabilities of the contents of the vectors in JBLK, in this case PC1, and added to the upper and lower bounds of the upper link, PBLK(0). The disjointed subproblem probability calculated above is added to each of these upper and lower bounds to give the system estimates for this level. If the difference between these bounds has reached the accuracy required, we quit; otherwise, we clear the bounds values from all PBLK entries and proceed to level 3.

The first or leftmost subproblem may be disjointed. In this case the components are all equal and C2 is chosen. The "right-hand" subproblem consists of only one simple set so its probability is entered in the DP portion of its PBLK entry. The "left-hand" subproblem is not yet disjoint, so it is continued to the next level and its bounds are calculated and placed in the appropriate entries of the PBLK entry. The remaining two subproblems of this level have no components in common, so their probabilities can be calculated and summed in the DP portion of their PBLK entries. The system probability can be calculated as before.

Level 5 (not shown on the diagram) would consist of only one subproblem which would have no components in common, and its solution would complete the disjointing process, if necessary.

BEST COPY AVAILABLE

References

1. G. C. Corynen, *Evaluating the Response of Complex Systems to Environmental Threats: the Sigma Pi Method*, Lawrence Livermore National Laboratory, Livermore, CA, UCRL-53399 (May 1983).

2. R. M. Thatcher, *Evaluating the Effects of Seismic Events on Systems in a Nuclear Facility Using the Sigma Pi Method*, Lawrence Livermore National Laboratory, Livermore, CA, UCID-19810 (May 1983).

3. G. C. Corynen, *'STOP': A Fast Procedure for the Exact Computation of the Performance of Complex Probabilistic Systems*, Lawrence Livermore National Laboratory, Livermore, CA, UCRL-53230 (1982).

BEST COPY AVAILABLE

APPENDIX A
Moving SIGPI to Other Computers

This section contains compile module information, parameter definitions, overlay segment definitions, and notes on word sizes, REAL and BOOLEAN variables, and possible use of library functions.

A.1. Compile Modules

The SIGPI source code comes in several compile modules. The modules roughly parallel the overlay segments, but do not contain the same routines exactly. The only optional modules are the PDEP/DPDEP ones. Use DPDEP when double precision REAL numbers are needed. The compile modules are:

Module PRE — Preprocessor.

Routines:

PREPROS — Reads option line and sets flags, reads set data, and loads the first set of probability data.

Programs called: DEPREAD, FILL, GETSYM, PSET, VSEEK.

Called by: SIGPI.

PSET — Reads in set data and calculates first set of subproblem tables.

Programs called: CLNA, LVSUM, VSEEK.

Called by: PREPROS.

DEPREAD — Reads in mean and covariance data and calculates individual probabilities for components.

Programs called: LVSUM, PFF, VSEEK.

Called by: PREPROS.

Module DPDEP — Double precision version of routines to calculate the probability of a component given its mean and standard deviation.

Routines:

PFF — Calculates probability of a component, using the standard error function.

Programs called: DERFC

Called by: DEPREAD, DEPUP.

BEST COPY AVAILABLE

DERFC — Calculates the double precision error function.
Programs called: D1MACH, DCSEVL, INITDS.
Called by: PFF.

D1MACH — Defines constants for many different computers. Modify the comment lines to reflect the computer being used.
Programs called: none.
Called by: DERFC.

DCSEVL — Calculates the double precision Chebyshev series.
Programs called: none.
Called by: DERFC.

INITDS — Initializes the Chebyshev series.
Programs called: none.
Called by: DERFC.

Module PDEP — Single precision version of routines to calculate the probability of a component given its mean and standard deviation.

Routines:

PFF — Calculates the probability of a component using the standard error function.
Programs called: ERFC
Called by: DEPREAD, DEPUP.

ERFC — Calculates a single precision error function.
Programs called: S1MACH, CSEVL, INITS.
Called by: PFF.

R1MACH — Defines constants for many different computers. Modify the comment lines to reflect the computer being used.
Programs called: none.
Called by: ERFC.

CSEVL — Calculates the Chebyshev series.
Programs called: none.
Called by: ERFC.

INITS — Initializes the Chebyshev series.
Programs called: none.
Called by: ERFC.

BEST COPY AVAILABLE

Module: SSIG — The main program module and main subroutines for Sigma processing of independent coherent data, and Pi processing of all independent data.

Routines:

SIGPI — Is the main program.

Programs called: PREPROS, MAINLOOP, COMPLOOP, DMLOOP

Called by: none

MAINLOOP — Does Sigma processing of coherent, independent data.

Programs called: BLKTEST, CLNA, DOM, DUMP, ESTIMATE, LNCHECK, LOAD, LOADCUR, LVSUM, OPT, PCHECK, PROD, REMOVE1, SPOOL, STORCUR, STORNEW, VSEEK, PROD

Called by: SIGPI

LNCHECK — Checks for special case where subproblems have no components in common. The function returns a TRUE value if that case is found.

Programs called: MAXA, PROD.

Called by: MAINLOOP, SPOOL.

PCHECK — Calculates disjoint probability if subproblem is a single set; otherwise, calculates upper and lower bounds.

Programs called: LVSUM, PROD.

Called by: BLKTEST, MAINLOOP, SPOOL.

ESIMATE — Calculates system upper and lower bounds at the end of each level.

Programs called: PROD

Called by: COMPLOOP, MAINLOOP, SPOOL

PROD — Calculates the product of probabilities of components in a vector.

Programs called: none

Called by: CLNCHCK, COMPLOOP, CPCHECK, CREMOVE1, DLNCHECK, ESIMATE, JBLKUP, JMIX, LNCHECK, MAINLOOP, PCHECK, REMIX, REMOVE1, SMIX, SPOOL.

BLKTEST — Does partitioning for independent, coherent data. The function returns a TRUE value if the last partition consists of a single set.

BEST COPY AVAILABLE

Programs called: CLNA, PCHECK, LV3UM, REMOVE1, STORNEW, VSEEK.
Called by: MAINLOOP.

REMOVE1 Removes components which have the same value in all sets.
Programs called: CLNA, PROD, EQVEC, LVSUM.
Called by: BLKTEST, MAINLOOP.

OPT Finds optimum J.
Programs called: EQVEC, FILL, FIRSTJ, LVSUM, MAXA, MAXV, MINV, REMOVE2, VSEEK.
Called by: MAINLOOP.

REMOVE2 Tests for identical optimum J's.
Programs called: EQVEC, LVSUM.
Called by: OPT.

DOM Looks for sets to be absorbed.
Programs called: none.
Called by: DMPROC, MAINLOOP.

SPOOL Does Pi processing.
Programs called: CLNCHECK, ESTIMATE, CPCHECK, GETNEXT, DUMP, LNCHECK, LOAD, PCHECK, PROD, VSEEK.
Called by: COMPLOOP, MAINLOOP.

GETNEXT Loads next subproblem for SPOOL.
Programs called: DUMP, LOAD, LOADCUR.
Called by: SPOOL.

Module: CSIG — Routines for Sigma processing of independent noncoherent data.

Routines:

COMPLOOP Does Sigma processing of independent, noncoherent data.
Programs called: CBLKTEST, CDOM, CLNA, CLNCHECK, COPT, CPCHECK, CREMOVE1, DUMP, ESTIMATE, LOAD, LOADCUR, LVSUM, PROD, SPOOL, STORCUR, STORNEW, VSEEK.
Called by: SIGPI.

BEST COPY AVAILABLE

CLNCHECK Checks for special case of subset with no components in common (noncoherent version). The function returns a TRUE value if that case is found.
Programs called: MAXA, PROD.
Called by: COMPLOOP, SPOOL.

CPCHECK Calculate disjoint probability if subproblem is a single set; otherwise, calculates upper and lower bounds for subproblem.
Programs called: PROD.
Called by: CBLKTEST, COMPLOOP, SPOOL.

CBLKTEST Does partitioning for noncoherent data.
Programs called: CLNA, CPCHECK, CREMOVE1, LVSUM, STORNEW, VSEEK.
Called by: COMPLOOP.

CREMOVE1 Tests for any component having the same value for all sets, and removes it from subproblem.
Programs called: CLNA, EQVEC, LVSUM, PROD.
Called by: CBLKTEST, COMPLOOP.

COPT Finds optimum J for noncoherent data.
Programs called: CREMOVE2, EQVEC, FILL, FIRSTJ, LVSUM, MAXA, MAXV, MINV, VSEEK.
Called by: COMPLOOP.

CREMOVE2 Generates a vector of all components having the same value as optimum J for all sets in subset.
Programs called: EQVEC, LVSUM.
Called by: COPT.

CDOM Tests for absorption (noncoherent version).
Programs called: none.
Called by: COMPLOOP, DCPROC.

Module: SD — Routines for Sigma processing of dependent coherent data and common routines for dependent coherent and noncoherent data.

Routines:

DMLOOP Schedules Sigma processing for dependent data. (The equivalent process is built into MAINLOOP and COMPLOOP for independent data.)
Programs called: DCPROC, DEST, DLOADCUR, DMPROC, DSPOOL, DUMP, LOAD.
Called by: SIGPI.

BEST COPY AVAILABLE

DMPROC Does Sigma processing for dependent coherent data.
Programs called: CLNA, DANAL, DBLKTEST, DLNCHECK, DOM, DOPT, DREMO, DREMOVE1, DSTORCUR, DSTORNEW, LVSUM, PLO, SETPROB, VSEEK.
Called by: DMLOOP.

SMIX Mixs a vector of components into a set of vectors defined by mutual dependencies.
Programs called: DPROB, LVLIST, PROD.
Called by: DCREMO, DREMO, PLO, REDOJS.

PLO Calculates lower bound for subproblem.
Programs called: FILL, LVSUM, SETPROB, SMIX.
Called by: DBLKTEST, DCBLKTST, DCPROC, DCREMO, DMPROC, DREMO, REMIX.

DEST Calculates system bounds at end of each level.
Programs called: SETPROB.
Called by: DMLOOP, DSPOOL.

DLNCHECK Tests for special case of subproblem with no components in common. This function returns a TRUE value if that case is found.
Programs called: DPROB, LVLIST, LVSUM, MAXA, PROD, SETPROB.
Called by: DMPROC, REMIX.

DANAL Analyzes vector of optimum J's for dependency problems, and inserts the vector into defined component vectors..
Programs called: DOPT, FILL, FIRSTJ, JMIX, LVLIST, LVSUM.
Called by: DMPROC.

JMIX Given a vector of optimum J's and a set of vectors defined by mutual dependencies, defines two new sets of vectors such that the optimum J's are inserted into the vectors of mutual dependency, and the optimum J's are defined to be "1" for the "right-hand" set and defined to be "0" for the "left-hand" set.
Programs called: DPROB, LVSUM, PROD.
Called by: DANAL, DCANAL.

DBLKTEST Does partition processing for dependent coherent data. This function returns a TRUE value only if partitioning takes place and the last partition consists of only one set.
Programs called: CLNA, DREMOVE1, DSTORNEW, LVLIST, LVSUM, PLO, VSEEK.
Called by: DMPROC.

BEST COPY AVAILABLE

DREMOVE1 Removes from a subproblem any component that always has a "don't care" value.
Programs called: EQVEC, LVSUM.
Called by: DBLKTEST, DMPROC.

DREMO Processes an optimum J that has the same value in all sets of the subproblem.
Programs called: CLNA, PLO, SETPROB, SMIX.
Called by: DMPROC.

DOPT Finds the optimum J component for dependent, coherent data.
Programs called: DREM2, EQVEC, FILL, FIRSTJ, LVSUM, MAXA, MAXV, VSEEK.
Called by: DANAL, DMPROC.

DREM2 Defines a vector of J's that are identical with the optimum J.
Programs called: EQVEC, LVSUM.
Called by: DOPT.

SETPROB Calculates the product of a set of probability values for defined mutually dependent vectors.
Programs called: none.
Called by: DCLNCHCK, DCREMO, DEST, DLNCHECK, DMPROC, DCPROC, DREMO, PLO, REMIX.

DPROB Calculates the probability of a vector of mutually dependent components.
Programs called: ADAPT, FILL, INVERT, LVLIST, LVSUM.
Called by: DCLNCHCK, DLNCHECK, JMIX, SMIX.

ADAPT Calculates the "n" dimensioned integral of FUNCTN.
Programs called: FUNCTN.
Called by: DPROB.

FUNCTN Probability function for the ADAPT integral.
Programs called: MATMUL.
Called by: ADAPT.

MATMUL Matrix multiply.
Programs called: none.
Called by: FUNCTN.

INVERT Matrix inversion.
Programs called: none.
Called by: DPROB.

BEST COPY AVAILABLE

Module: SDC — Sigma processing for dependent, noncoherent data.

Routines:

DCPROC — Does Sigma processing for dependent, noncoherent data.

Programs called: CDOM, CLNA, DCANAL, DCBLKTST, DCLNCHCK, DCOPT, DCREM0, DCREMOV1, DSTORCUR, DSTORNEW, LVSUM, PLO, SETPROB, VSEEK.

Called by: DMLOOP.

DCLNCHCK — Checks for special case of subproblem with no components in common. This function returns a TRUE value if that case is found.

Programs called: DPROB, LVLIST, LVSUM, MAXA, PROD, SETPROB.

Called by: DCPROC, REMIX.

DCANAL — Analyzes optimum J for dependency suitability.

Programs called: DCOPT, FILL, JMIX, LVLIST, LVSUM.

Called by: DCPROC.

DCBLKTST — Does partition processing for dependent, noncoherent data. This function returns a TRUE value only if partition takes place and the last partition consists of only a single set.

Programs called: CLNA, DCREMOV1, DSTORNEW, LVLIST, LVSUM, PLO, VSEEK.

Called by: DCPROC.

DCREMOV1 — Removes components from subproblem that always have a "don't care" value.

Programs called: EQVEC, LVSUM.

Called by: DCBLKTST, DCPROC.

DCREM0 — Processes vector of optimum J's that have the same value in all sets of the subproblem.

Programs called: CLNA, PLO, SETPROB, SMIX.

Called by: DCPROC.

DCOPT — Finds optimum J for dependent, noncoherent data.

Programs called: DCREM2, EQVEC, FILL, FIRSTJ, LVSUM, MAXA, MAXV, VSEEK.

Called by: DCANAL, DCPROC.

DCREM2 — Generates vector of J's that are identical to optimum J in all sets of the subproblem.

Programs called: EQVEC, LVSUM.

Called by: DCOPT.

BEST COPY AVAILABLE

Module: SDP — Pi processing for dependent data.
Routines:

DSPOOL — Does Pi processing for dependent data.
Programs called: DEPUP, DEST GETNEXTD, JBLKUP, LOAD, REMIX, VSEEK.
Called by: DMLOOP.

GETNEXTD — Gets next subproblem for DSPOOL.
Programs called: DLOADCUR, DUMP, LOAD.
Called by: DSPOOL.

DEPUP — Gets new covariance and mean data for components and makes a vector of all components that have changed dependencies.
Programs called: LVSUM, PFF, VSEEK.
Called by: DSPOOL.

REMIX — Calculates new bounds for each subproblem.
Programs called: CLNA, DCLNCHCK, DLNCHECK, DSTORCUR, LVSUM, PLO, PROD, REDOJS, SETPROB.
Called by: DSPOOL.

JBLKUP — Recalculates the JBLK array for changes in mutual dependencies and new probability values.
Programs called: PROD, REDOJS.
Called by: DSPOOL.

REDOJS — Regenerates vectors of mutual dependency.
Programs called: LVSUM, SMIX.
Called by: JBLKUP, REMIX.

Module: SIO — Common routines to handle subproblem memory and disk storage.
Routine:

ROOM — Finds room in memory or on disk for subproblem.
Programs called: DUMP.
Called by: DSTORCUR, DSTORNEW, STORCUR, STORNEW.

DUMP — Dumps memory subproblem area onto temporary disk file.
Programs called: LARGER.
Called by: ROOM, MAINLOOP, COMPLOOP, DMLOOP, GETNEXT, SPOOL.

BEST COPY AVAILABLE

LARGER Generates new temporary file when old one fills up.
Programs called: ADDONE.
Called by: DUMP.

ADDONE Generates name for next temporary file.
Programs called: none.
Called by: LARGER, LOAD.

LOAD Loads block of data from temporary file to memory.
Programs called: ADDONE.
Called by: COMPLOOP, DMLOOP, DSPOOL, GETNEXT, GETNEXTD, MAINLOOP, SPOOL.

Module: FUN Replaced library functions. See section A.7 for definitions.

Routines:

BMASK (block data), CLNA, STORCUR, STORNEW, LOADCUR, DSTORCUR, DSTORNEW, DLOADCUR, LVSUM, LVLIST, VSEEK, MAXA, MAXV, MINV, FIRSTJ, EQVEC, FILL, GETSYM.

COPY AVAILABLE

A.2. Parameters

Several parameters define the sizes of the arrays and other critical definitions for each installation. Change the parameters with a text editor to meet the needs of each installation. Take care to ensure identical definitions in each module of the source code.

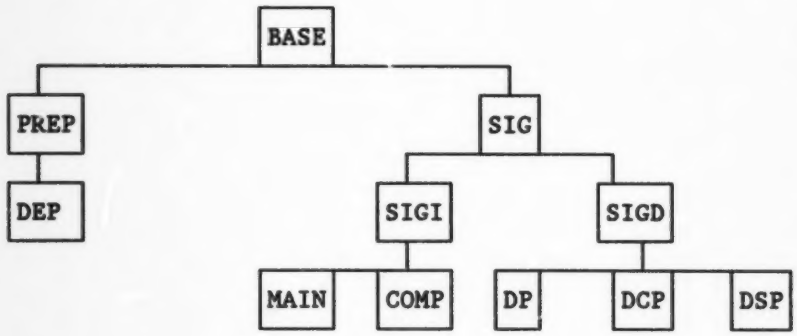
Parameter	Defined Value	Definition
WDSZ	64	Number of bits per INTEGER word.
NCOMP	256	Maximum number of components which can be processed.
NSETS	2000	Maximum number of simple sets which can be processed.
NWD	NCOMP/WDSZ	Maximum number of words per bit vector.
TFILESZ	500	Number of records per file to use for overflow data.
RECSZ	1664	Number of words per record for temporary files.
ENDMEM	20*RECSZ	Size of subproblem storage in memory.
TTYO	6	Unit number for terminal output file.
TTYI	6	Unit number for terminal input.
LPBLK	200	Maximum number of entries in the PBLK array. This number determines themaximum number of partitions that can be handled.
LJB	200	Maximum number of entries in the JBLK array. If JBLK size is not sufficient, no further partitions will be allowed. JBLK and PBLK do not necessarily grow at the same rate. With independent variables, PBLK grows faster; with dependent variables, JBLK grows faster.
JSMAX	20	Maximum number of groups of mutually dependent components in each disjointed subproblem.

BEST COPY AVAILABLE

Appendix A

A.3. Overlays

For systems that use overlays, the following tree structure is the most desirable. Note that some of the routines are duplicated for most efficient access; they are marked with an asterisk.



With the segments defined as follows:

Segment	Routine	Comments
BASE		Common routines for all overlays.
	SIGPI	SIGPI main driver.
	CLNA	Calculates LNA and LNA1 arrays (see function definitions below).
	LVSUM	Bit count function.
	LVLIST	Bit list function.
	VSEEK	Variable seek.
	FILL	Fill an array with a scaler.
PRE	PREPROS	Preprocessor overlay.
	GETSYM	Gets next symbol from input line.
	PSET	Reads input cut sets.
DEP	DEPREAD	Dependent data input routine.
	PFF*	Calculates single probability for each component.
	ERFC*	Complementary error function.
	R1MACH*	Defines machine-dependent constants for ERFC.
	CSEVL*	Evaluate Chebyshev series for ERFC.
	INITS*	Initialize for CSEVL.

---

* Routines appearing in more than one segment.

BEST COPY AVAILABLE

Segment	Routine	Comments (cont.)
SIG		Common routines for all SIGMA processing.
	ROOM	Determines if there is room for new subproblem in memory storage, otherwise call dump routine.
	DUMP	Writes memory subproblem storage area to temporary disk file.
	LARGER	Assigns another disk file if temporary disk is full.
	ADDONE	Function that creates a new name for temporary disk files.
	LOAD	Gets subproblems from temporary disk storage to memory subproblem storage area.
	MAXA	Find MAX entry in array.
	MAXV	Find MAX entry in array under control of bit vector.
	MINV	Find MIN entry in array under control of bit vector.
	FIRSTJ	Find first bit set in vector.
	EQVEC	Generate bit vector of all entries in array equal to a given value.
	PROD	Calculate product of probabilites of components in a simple set.
SIGI		Routines common to independent component processing.
	STORCUR	Store current subproblem in memory storage area.
	STORNEW	Store new subproblem in memory storage area.
	LOADCUR	Load subproblem from memory storage area.
	ESTIMATE	Calculate upper and lower bounds for system.
	SPOOL	Load new conditional data set and recalculate system probabilities.
	GETNEXT	Load next subproblem for SPOOL.
MAIN	MAINLOOP	Main SIGMA processing for coherent independent components.
	BLKTEST	Partitioning processor.
	REMOVE1	Search for components that are all "0", "1", or "don't care".
	LNCHECK	Check for special case of subproblem with no components in common.
	OPT	Find optimum components for disjointing.
	REMOVE2	Test for multiple identical components that satisfy the optimum test.
	PCHECK	Calculate disjoint subproblem probability or bounds for subproblems not yet disjoint.
	DOM*	Dominance test for absorption of sets.

BEST COPY AVAILABLE

Segment	Routine	Comments (cont.)
COMP	COMPLOOP	Main SIGMA processing for noncoherent independent components.
	CBLKTEST	Partitioning processor.
	CREMOVE1	Search for components that are all "0", "1", or "don't care".
	CLNCHECK	Check for special case of subproblem with no components in common.
	COPT	Find optimum components for disjointing.
	CREMOVE2	Test for multiple identical components that satisfy the optimum test.
	CPCHECK	Calculate disjoint subproblem probability or bounds for subproblems not yet disjoint.
	CDOM*	Dominance test for absorption of sets.
SIGD	DMLOOP	Main control and scheduling for dependent data processing. This overlay also contains all of the common routines for the two dependent data overlays.
	DSTORCUR	Store current subproblem in memory storage area.
	DSTORNEW	Store new subproblem in memory storage area.
	DLOADCUR	Load subproblem from memory storage area.
	PLO	Calculate lower bound for subproblem.
	SETPROB	Calculate product of probabilities of groups of mutually dependent components.
	JMIX	Given a component J and a set of groups of mutually dependent components already specified as "0" or "1", generate two new sets with J specified in the proper group as "0" in one and as "1" in the other.
	SMIX	Given bit vectors of components specified as "0" or "1" and a set of groups of mutually dependent components already specified, insert the new vector components into the proper groups.
	DEST	Calculate system probability from probabilities of partitions.
	DSPOOL	Load next conditional data set. (Note that all other subroutines for processing additional conditional data sets are in a separate overlay, but this one must be here to call DEST.)
	DPROB	Calculate probability of mutually dependent components.
	ADAPT	N dimensioned integral to do calculation for DPROB.
	INVERT	Calculate matrix inversion for DPROB.
	FUNCTN	Function used by integral.
	MATMUL	Matrix multiply used by FUNCTN.

BEST COPY AVAILABLE

Segment	Routine	Comments (cont.)
DP	DMPROC	Main SIGMA processor for dependent coherent data.
	DBLKTEST	Partitioning processor.
	DLNCHECK	Test for special case of subproblem with no components in common.
	DREMOVE1	Test for any component that has "don't care" value in all sets of subproblem.
	DOPT	Find optimum component to disjoint subproblem.
	DREM2	Test for identical components in subproblem that meet DOPT requirements.
	DREM0	Process special case in which optimum component always has "0" or "1" value.
	DANAL	Analyze optimum component(s) for suitability of disjointing, by considering mutual groups of dependent components.
	DOM*	Dominance test for absorption of sets.
DPC	DCPROC	Main SIGMA processor for dependent noncoherent data.
	DCBLKTST	Partitioning processor.
	DCLNCHCK	Test for special case of subproblem with no components in common.
	DCREMOV1	Test for any component that has "don't care" value in all sets of subproblem.
	DCOPT	Find optimum component to disjoint subproblem.
	DCREM2	Test for identical components in subproblem that meet DCOPT requirements.
	DCREM0	Process special case in which optimum component always has "0" or "1" value.
	DCANAL	Analyze optimum component(s) for suitability of disjointing, by considering mutual groups of dependent components.
	CDOM*	Dominance test for absorption of sets.
DSP		Overlay for processing additional conditional data sets for dependent data.
	DEPUP	Read in next conditional dependent probability data set.
	JBLKUP	Update groups of mutually dependent component vectors in the JBLK array, by considering the new dependencies.
	GETNEXTD	Get one subproblem at a time for reprocessing.
	REMIX	Process the groups of mutually dependent component vectors in a subproblem, by considering the new dependencies.
	PFF*	Calculates single probability for each component.

---

* Routines appearing in more than one segment.

BEST COPY AVAILABLE

Segment	Routine	Comments (cont.)
	ERFC*	Complementary error function.
	R1MACH*	Defines machine-dependent constants for ERFC.
	CSEVL*	Evaluate Chebyshev series for ERFC.
	INITS*	Initialize for CSEVL.

---

* Routines appearing in more than one segment.

A.4. Word Size Considerations.

The parameter WDSZ defines the number of bits per word as defined for an integer variable on the computer system being used. This is sufficient for most instances where the word size is critical, but some other changes may be needed in the size and definitions of the bit masks (see BMASK in A.7, system functions below) and in the temporary storage handling routines (see A.5, Real Variable Considerations).

A.5. Real Variable Considerations.

SIGPI was designed on a CRAY-1 computer with a 64-bit word size, giving approximately 16 digits of accuracy. To move to a mini-computer with 32-bit word sizes or less, it would be desirable to use double precision real variables to hold the probability values. Use a text editor to change all REAL declarations to REAL*8. Constants that are used by the program should be changed from the nE1.0 format to the nD1.0 format. The program module that does the single variable probability calculation for dependent variables has been included in two versions, PDEP for single precision and DPDEP for double precision. Additionally, the routines that load and store the current subproblems into the subproblem storage block must be modified to allow for the differences in word sizes between REAL*8 variables and INTEGER variables, because all data is handled as INTEGER in these routines. The routines (D)LOADCUR, (D)STORCUR, and (D)STORNEW have commented areas defining these modifications.

A.6. Boolean Operations

Full word boolean operations .AND., .OR., .NOT. must be available.

BEST COPY AVAILABLE

A.7. Library Functions.

Many advanced computer systems have included a variety of special routines in their libraries to take advantage of the special features of their computers. The routines generally contained in the FUN module of subroutines are replacements for such special routines and can probably be replaced by equivalent fast versions provided by the computer manufacturer. The following list defines the purpose of the FUN module routines.

BLOCK DATA BMASK -a predefined list of masks to address each bit of a computer word starting with the leftmost. This may be replaced with a fast mask manufacture function if one is available.

SUBROUTINE CLNA(I,SET,LN,JA,IA) - a subroutine to count the occurrences of each component in an array of component vectors. SET is the matrix of bit vectors, either ISSET or ICSET. IA is an array of indices of the bit vectors in SET to be counted, and I is the length of the array. JA contains a bit vector defining which bits are to be counted and will be either JALPHA or JANEW. The entries in the array LN (either LNA or LNA1) will be set equal to the count of the bits defined by JA, in the sets defined by SET; other entries in LN will be zero.

I = LVSUM (VEC, CT) - function that gives a count of the "1" bits in VEC for CT bits.

I = LVLIST (TABLE, VEC, CT) - function that gives a list of the "1" bits in VEC for CT bits. TABLE is an array to receive the list, at least CT words long. The bits in the vector are defined starting with the leftmost bit of the first word as number one.

CALL VSEEK (POS, ARRAY, NUM, CT) - subroutine that looks for an entry in ARRAY that is equal to NUM value. POS contains the index if one is found or zero if one is not found. CT is the array size.

CALL MAXA (MAXN, INDX, ARRAY, CT) - subroutine that finds the maximum entry in ARRAY. MAXN contains the value of the maximum entry, and INDX contains its index. CT is the array size.

CALL MAXV (MAX, INDX, ARRAY, CT, VEC) - subroutine that finds the maximum entry in ARRAY using VEC as a control vector (only those entries in ARRAY with have a corresponding bit set in VEC will be considered). INDX is the index of the maximum value found, and CT is the number of words to be considered in ARRAY and the number of bits in VEC.

BEST COPY AVAILABLE

Appendix A

CALL MINV (MINN, INDX, ARRAY, CT, VEC) - subroutine that finds the minimum entry in ARRAY using VEC as a control vector (only those entries in ARRAY which have a corresponding bit set in VEC will be considered). INDX is the index of the minimum value found, and CT is the number of words to be considered in ARRAY and the number of bits in VEC.

I = FIRSTJ (VEC, CT) - function to find the first or leftmost "1" bit set in bit vector VEC. CT is the number of bits in VEC. The index returned counts from the leftmost bit of the first word as one.

CALL EQVEC (VEC, ARRAY, NUM, CT) - subroutine that searches ARRAY for NUM value and generates a control bit vector, i.e., the corresponding bit in VEC is set to 1 where ARRAY equals NUM. CT is the array size.

CALL FILL (ARRAY, NUM, CT) - Broadcasts NUM into ARRAY for CT words.

STRING = GETSYM (LINE,DEL,N) - Gets a character STRING from a character array LINE using an array of N delimiters in DEL. Leading delimiters are removed. Characters up to but not including the next delimiter are placed in STRING and removed from LINE.

EST COPY AVAILABLE

APPENDIX B
Test Data

This appendix contains data sets and expected answers to use in verification of the SIGPI program. There are four sets of cut set data: a very small and a larger set each of coherent and noncoherent data. There is one set of probability data assuming statistical independence, and one set of mean and covariance data for adding some statistically dependent components to the calculation. These files should be included in the package with the SIGPI source code.

B.1. Cut Set Data

The cut set data in the following files was randomly generated and represents a worse-case set for processing. Sets generated by fault-tree processing of models would have much more structure and would be processed much more quickly. The following data representation needs to be converted to an unformatted file for processing by SIGPI. The first number on each record is the number of elements in the corresponding cut set. Noncoherent data has a minus sign. For this document, records which do not fit conveniently on a line are continued with indentations on the following line.

B.1.1. Small Coherent Data Set

File TEST1

Size	Components in set							
4	14	21	4	20				
8	6	12	16	20	3	7	15	19
5	13	17	3	15	19			
3	12	22	18					
2	14	5						

B.1.2. Small Noncoherent Data Set

File TEST2

Size	Components in set							
4	14	21	-4	-20				
8	6	12	16	20	-3	-7	-15	-19
5	13	17	-3	-15	-19			
3	12	22	-18					
2	14	-5						

BEST COPY AVAILABLE

Appendix B

B.1.3. Large Coherent Data Set

File TEST3

Size	Components in set											
12	14	21	26	32	36	40	4	20	23	27		
	35	39										
8	13	17	32	42	3	15	19	38				
10	14	26	29	36	40	5	25	28	30	39		
11	10	18	20	24	27	7	14	19	23	25	30	
6	5	9	31	8	24	33						
4	19	29	22	40								
7	7	18	33	38	14	31	34					
6	9	25	38	3	18	30						
14	5	16	20	24	27	34	42	4	14	18	21	25
	31	37										
8	22	26	30	32	15	24	29	31				
5	21	37	18	28	39							
7	3	5	16	33	4	13	21					
6	16	18	27	7	17	20						
5	22	31	17	23	37							
8	4	18	30	36	9	19	32	39				
12	9	15	24	29	32	37	10	17	26	31	36	42
7	8	21	39	41	10	33	40					
5	23	36	8	24	40							
10	4	11	21	32	39	7	14	27	36	42		
7	14	26	34	39	19	32	36					

B.1.4. Large Noncoherent Data Set

File TEST4

Size	Components in set										
7	14	21	-26	-4	20	23	27				
5	13	17	3	15	19						
6	14	26	29	5	25	28	30				
11	10	18	-20	24	27	7	14	-19	23	25	30
5	-5	9	31	8	24						
3	19	29	22								
4	7	-18	-14	31							
5	9	25	-3	18	-30						
10	5	16	20	24	27	4	14	18	21	25	
6	22	26	30	15	24	29					
3	21	18	28								
6	3	5	16	4	13	-21					
6	16	18	27	7	17	20					
4	22	31	17	23							
5	4	-18	30	9	-19						
7	9	15	-24	-29	10	17	-26				
3	8	21	10								
3	23	8	24								
6	4	11	21	7	14	27					
3	-14	26	-19								

BEST COPY AVAILABLE

B.2. Independent Probability Data

The following file contains three sets of independent probability data that can be used with any of the four cut set files above.

File PTEST

Component	Probability
3	.5
6	.5
8	.5
9	.5
11	.5
12	.5
13	.5
14	.5
15	.5
16	.5
17	.5
18	.5
19	.5
20	.5
21	.5
22	.5
23	.5
24	.5
25	.5
26	.5
27	.5
28	.5
29	.5
30	.5
31	.5
32	.5
33	.5
34	.5
35	.5
36	.5
37	.5
38	.5
39	.5
40	.5
41	.5
42	.5
43	.5
*****	
3	.1
6	.1
8	.1
9	.1
11	.1
12	.1
13	.1
14	.1

BEST COPY AVAILABLE

Appendix B

File PTEST (Cont.)

```
  15      .1
  16      .1
  17      .1
  18      .1
  19      .1
  20      .1
  21      .1
  22      .1
  23      .1
  24      .1
  25      .1
  26      .1
  27      .1
  28      .1
  29      .1
  30      .1
  31      .1
  32      .1
  33      .1
  34      .1
  35      .1
  36      .1
  37      .1
  38      .1
  39      .1
  40      .1
  41      .1
  42      .1
  43      .1
*****
   3      .5
   6      .5
   8      .5
   9      .5
  11      .5
  12      .5
  13      .5
  14      .5
  15      .5
  16      .5
  17      .5
  18      .5
  19      .5
  20      .5
  21      .5
  22      .5
  23      .5
  24      .5
  25      .5
  26      .5
  27      .5
  28      .5
  29      .5
```

BEST COPY AVAILABLE

```
File PTEST (Cont.)
  30      .5
  31      .5
  32      .5
  33      .5
  34      .5
  35      .5
  36      .5
  37      .5
  38      .5
  39      .5
  40      .5
*****
```

B.3. Mean and Covariance Data Set

The following data file contains two sets of mean and covariance data for components 4, 5, 7, and 10. The first record of each set contains the number of components in the set. The next group of records contains the component number and its statistical mean. The last group of each set contains the covariance matrix for those components. The second set follows immediately after the first. The final zero marks the end of the data sets. The data has been compressed slightly to fit the document page.

```
          File DTEST
3
   4    1.2370236452e+00
   5    2.4055484815e+00
   7    0.5190527407e+00
2.7819240252e-01  1.8080697096e-01  1.8012193028e-01
1.8080697096e-01  2.4650009527e-01  1.6642897320e-01
1.8012193028e-01  1.6642897320e-01  2.0010247397e-01
4
   4    1.1370236452e+00
   5    1.4055484816e+00
   7    0.5190527407e+00
  10    1.2000000000e+00
2.5819240252e-01  1.5080697096e-01  0.0         1.2e-01
1.5080697096e-01  2.4650009527e-01  0.0         6.0e-02
0.0               0.0               1.1         0.0
1.2e-01           6.0e-02           0.0         1.4e-01
0
```


BEST COPY AVAILABLE

Appendix B

B.4. Computer Runs Using the Data Sets

The following combinations of data are recommended to verify correct operation of SIGPI. Input to SIGPI is followed by the expected answers.

B.4.1. Small set - coherent, independent data.

```
SIGPI
>
I-TEST1,P-PTEST,ACC-2

     Upper Bound        Lower Bound
      0.1523E+00         0.1523E+00
```

B.4.2. Small set - noncoherent, independent data.

```
SIGPI
>
I-TEST2,P-PTEST,ACC-2

     Upper Bound        Lower Bound
      0.5786E+00         0.5776E+00
```

B.4.3. Small set - coherent, dependent data.

```
SIGPI
>
I-TEST1,P-PTEST,D-DTEST,ACC-2

     Upper Bound        Lower Bound
      0.1546E+00         0.1536E+00
```

B.4.4. Small set - noncoherent, dependent data.

```
SIGPI
>
I-TEST2,P-PTEST,D-DTEST,ACC-2

     Upper Bound        Lower Bound
      0.5781E+00         0.5781E+00
```


BEST COPY AVAILABLE

B.4.5. Large set - coherent independent data, three conditions

```
SIGPI
>
I-TEST3,P-PTEST,ACC-2,CON-3

    Upper Bound          Lower Bound
     0.1667E+00           0.1654E+00
     0.1311E-03           0.1311E-03
     0.1667E+00           0.1654E+00
```

B.4.6. Large set - noncoherent independent data, three conditions.

```
SIGPI
>
I-TEST4,P-PTEST,ACC-2,CON-3

    Upper Bound          Lower Bound
     0.5117E+00           0.5068E+00
     0.8476E-01           0.8476E-01
     0.5117E+00           0.5068E+00
```

B.4.7. Large set - coherent dependent data, three conditions

```
SIGPI
>
I-TEST3,P-PTEST,D-DTEST,ACC-2,CON-3

    Upper Bound          Lower Bound
     0.1715E+00           0.1706E+00
     9.4414E-04           1.3454E-04
     0.2077E+00           0.1770E+00
```

B.4.8. Large set - noncoherent dependent data, two conditions

```
SIGPI
>
I-TEST4,P-PTEST,D-DTEST,ACC-2,CON-2

    Upper Bound          Lower Bound
     0.5254E+00           0.5224E+00
     0.1085E+00           0.1076E+00
```


BEST COPY AVAILABLE

NRC FORM 335 (2-84) NRCM 1102, 3201, 3202

U.S. NUCLEAR REGULATORY COMMISSION

BIBLIOGRAPHIC DATA SHEET

SEE INSTRUCTIONS ON THE REVERSE

1. REPORT NUMBER (Assigned by TIDC, add Vol. No., if any)
NUREG/CR-4800
UCID-20679

2. TITLE AND SUBTITLE
SIGPI: A User's Manual for Fast Computation of the Probabilistic Performance of Complex Systems

3. LEAVE BLANK

4. DATE REPORT COMPLETED
MONTH: September YEAR: 1986

5. AUTHOR(S)
C. J. Patenaude

6. DATE REPORT ISSUED
MONTH: May YEAR: 1987

7. PERFORMING ORGANIZATION NAME AND MAILING ADDRESS (Include Zip Code)
Lawrence Livermore National Laboratory
7000 East Avenue
Livermore, CA 94550

8. PROJECT/TASK/WORK UNIT NUMBER

9. FIN OR GRANT NUMBER
A0392

10. SPONSORING ORGANIZATION NAME AND MAILING ADDRESS (Include Zip Code)
Division of Reactor System Safety
Division of Nuclear Regulatory Research
U.S. Nuclear Regulatory Commission
Washington, DC 20555

11a. TYPE OF REPORT
Technical

b. PERIOD COVERED (Inclusive dates)

12. SUPPLEMENTARY NOTES

13. ABSTRACT (200 words or less)

The SIGPI program computes the probability of complex systems as defined by cut sets or other binary product sets. The SIGPI program uses two fast complementary methods of computing the probabilistic performance of complex systems: the II method and the Σ method.

The II algorithm exploits the fact that carefully defined system components are often statistically independent conditional to the environment in which they are embedded. The Σ algorithm computes the probability of combinations of components produced by the II algorithm by disjointing and partitioning such components, thereby allowing the exact computation of performance.

The program assumes input of up to three data types: cut set data in disjoint normal form, basic component probabilities for independent basic components, and/or mean and covariance data for statistically dependent basic components.

14. DOCUMENT ANALYSIS - a. KEYWORDS/DESCRIPTORS
probability calculations
probabilistic risk assessment
computer program
SIGPI computer codes

b. IDENTIFIERS/OPEN-ENDED TERMS

15. AVAILABILITY STATEMENT
Unlimited

16. SECURITY CLASSIFICATION
(This page) Unclassified
(This report) Unclassified

17. NUMBER OF PAGES

18. PRICE

66 BEST COPY AVAILABLE

U.S. G.P.O. 1987-181-682:60099